Theme 1

A Program of Inquiry

by

Jon Awbrey
Susan Awbrey

Introduction

Theme 1 is a computer program with a dual objective: to perform tasks of inductive and deductive problem solving that arise in research, and to explore the integration of these reasoning types in human and computer information processing. Theme 1, the application program, is both the instrumental means and the implemented result of advances made in the corresponding program of research.

In pursuit of these complementary aims, it is necessary to choose problem domains in practice that are simple enough to make progress on, generic enough to gain theoretical insight from, and yet both complex and concrete enough to have real applications. Two problem domains were selected as meeting these criteria.

As a task for inductive reasoning, Theme 1 addresses the problem of learning arbitrary formal languages specified at the two levels of words and sentences.

As a task for deductive reasoning, Theme 1 addresses the problem of modeling arbitrary formulas in the propositional calculus, or of finding normal forms.

Further explanation of these terms is given below, in the context of their use and in the examples of applications.

Theme 1 is divided into three main operational sections:

Order serves merely as a place to begin and as a utility for accessing a pair of optional menu files.

Index operates as an empirical program, to induce a literal model of a language as encountered in actual experience.

Study operates as a reasoning program, to deduce a logical model of a universe as described in symbolic expressions.

The present documentation for Theme 1 is divided into the following chapters:

Chapter 1 defines the notion of a two-level formal language and tells in what way the Index function can be said to learn or induce such languages.

Chapter 2 depicts the notation we use for propositional calculus and shows how to represent simple deductive problems in a form that the modeling functions of Study can help to clarify.

Chapters 3 & 4 are “construction zones”, briefly sketching ideas and applications that are currently in the process of being developed.

Appendix A collects a sample of source materials for the philosophy of mind and methodology that we use.

Appendix B is a user manual for the Theme 1 program. The first part provides general information about the main interactive control structure of Theme 1, the labeled prompt. The next part describes the specific functions of Theme 1 in outline form, under the corresponding prompt headings.

Chapter 1

Index: the Language Learner

A two-level formal language is given to us when we know three things: the alphabet, the words, and the sentences.

The **alphabet** A is a finite set, usually something like { a, b, c, ... }; in our case it will be some subset of the keyboard characters, minus whatever special symbols are reserved for commands.

The **words**, as a set W, are some subset of A* , the set of all finite sequences of alphabet elements. This corresponds to the usual *string* data type of most programming languages.

The **sentences**, as a set S, are some subset of W* , the set of all finite sequences of words. We will usually refer to such arbitrary word sequences as *lists*, *paths*, or *strands*, saving the designation *sentence* for use in the more structured and semantical domains of natural language.

Generally speaking, when there is no presumption of meaning attached to the elements of W and S, we will call them *strings* and *strands*, respectively. In what follows, the elements of W will usually be associated with meaningful features of experience or reasoning, and so may be thought of as *words*. In contexts where a suitable generalization applies or where the sense is clear, words and lists will be referred to indifferently as *sequences*.

We now consider the sense in which Index may be said to learn a two-level formal language, in other words, we discuss the particular kind of sequential modeling task that Index is designed to perform. In this design, we are seeking to isolate and to implement a fundamental (simple but not trivial) component of sequential learning that can be carried out effectively and efficiently in real time and under *empiricist* constraints, i.e., with minimal prior conceptual commitments or expectations built into the learner.

Imagine that the learner is presented with an environment consisting of a data stream like:

```
able baker can cook cakes  able baker can bake apples  ...
```

that is, a continuing sequence of elements from an alphabet A, with distinctly known delimiters indicating words and word sequences. For simplicity, let ends of words be marked with a single blank space and ends of word sequences with an extra blank, in addition to the one terminating the last word of the sequence. The text above has been marked in a corresponding fashion.

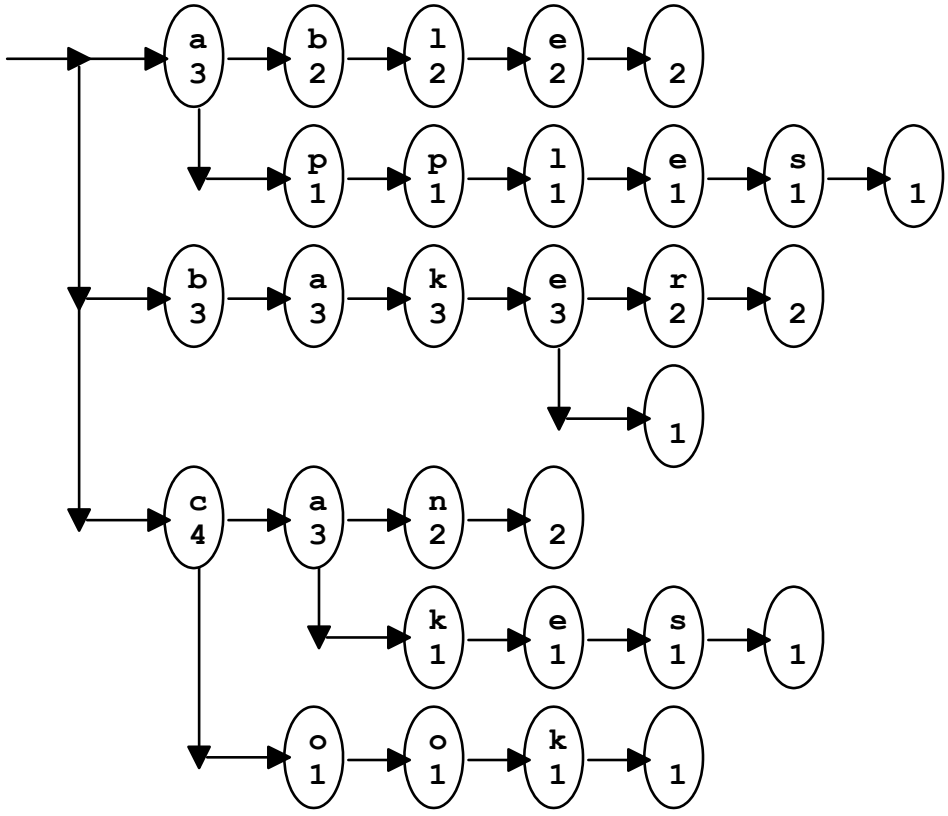
We consider that an agent (organism or program) has learned something of this environment when it has formed a model (a data structure) in its memory that prepares it for likely consequences of given partial sequences, based only the sequences that have actually occurred.

The subsets of W and S occurring in a finite initial portion of the data stream are necessarily some finite sets W′ and S′ and are therefore finite state languages in themselves. This is true whatever of the complexity of the unknown languages W and S that generate the data stream. This means that the current knowledge of the language represented by W′ and S′ at a given time t′ may be captured by a couple of finite state transition trees.

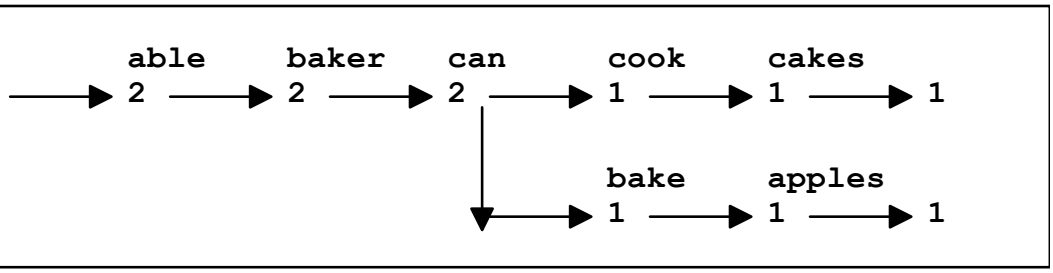
(a)

```
able baker can cook cakes  able baker can bake apples
```

(b)



(c)



(d)

```
able
  baker
    can
      cook
        cakes
      bake
        apples
```

Figure 1. Finite State Transition Trees

Examples of finite state transition trees are presented in Figure 1. Part (a) of the figure repeats the data stream fragment just seen. It is from this segment of the language L that the transition trees for Words and Strands are constructed. Part (b) shows the W′ tree for Words considered as sequences of alphabet elements. Ends of words are indicated by blank entries on terminal nodes of the tree. The numerical values that appear may be interpreted either as incidence or as transition frequencies. Part (c) shows the S′ tree for Strands considered as sequences of words. Ends of strands are marked by blank nodes. Numbers give incidence or transition frequencies, as before. Part (d) gives the S′ tree in *outline* form.

Index is essentially a program for constructing a couple of finite state transition trees, called the lexical and literal approximations W′ and S′ or the current summary L′ = (W′, S′) of the data stream language L = (W, S). Index grows these trees in real time interaction with a data stream, typed in by a user at the keyboard acting in the role of environment.

Index also keeps track of incidence frequencies with which elements are rooted in their tree contexts. These counts may be viewed as empirical transition frequencies and used to compute transition probabilities and information measures of uncertainty, or entropies, at each juncture of the tree.

Index has a Sort function, which does not perform a genuine sort operation on the data trees, due to its difficulty under interactive time constraints, but merely realigns tree paths in the directions of their maximal frequencies.

With these capabilities the Index learner / inducer is able to respond at new levels of interaction. By using its data model to anticipate likely sequels and display current projections to each increment of partial input, Index can both acquire and provide information more quickly in its interaction with the teacher / environment data stream. This new level of interaction is facilitated by modifying slightly the effect of acceptor symbols and by reserving some new command symbols for partial accept and reject signals at the keyboard.

At this point the pragmatic demands of efficient performance seem to ruin our plan to maintain empirical equanimity, and we are forced into a theoretical commitment. Using the species of tree that we do, one which unifies the common prefixes of sequences and thus captures only their initial regularities, amounts to a prejudice that limits the induced model in favor of always regressing subsequent events upon antecedent states as predictors.

Although it would be feasible and useful to extend the model, at least to cover suffix commonalities, no efficient extension of the current algorithm would appear able to exploit all potential laws of sequential redundancy. Indeed, even at the level of effectiveness, no finite state model can hope to acquire all the relevant and succinct generalizations of a properly more complex language. (Cf: Any basic text in formal language theory, e.g., [DDQ].) Nevertheless, for an adaptive agent under the press of evolving selection, we suggest that some model is better than none. It will be a task for the future to see how this kind of model might be used as a basis for building better theories of given environments.

Chapter 2

Study: the Universe Modeler

The notation we use for propositional calculus is derived from the system of Logical Graphs developed by Charles S. Peirce around the turn of the century [P1, P2, P3]. These ideas were extended by the work of G. Spencer Brown in his book Laws of Form [SpB]. The present notation develops this system of Logical Graphs only by an operation of reflection that reapplies its own founding principles to the received formulation.

These founding principles, or esthetic imperatives, present throughout Peirce's logical work, are:

> Try to reduce the number of primitive notions.
>
> Try to vary what has been held to be constant.

Our present aim draws us quickly past these reflections, but see if you can recognize for yourself how they issue in the system that follows.

In propositional calculus, as used here, our purpose will be to describe some universe of discourse in terms of a set of features or propositions that are interpreted as primitive. We will want the capacity to define terms in terms of each other, and more generally to impose constraints on the logical compatibility of terms. Given a set of such definitions and constraints we would like the facility of deducing their logical equivalents and consequences.

The fastest way to the point will be to assume the reader already knows some version of propositional calculus, that domain of reasoning depicted in Venn diagrams and in Truth tables, and to present what follows as nothing more than a variant notation for it. In this way we only need to show how to represent an adequate set of basic expressions in the new notation.

We take as terms the words in some set W. These are usually interpreted as referring to features of or propositions about elements in the universe of discourse, whether objects, events, situations, or whatever category is under discussion. Terms correspond to the labeled circles of Venn diagrams or the sentence letters of Truth tables.

We take such terms to be the simplest expressions of the calculus for W, and we form more complex expressions by combining any expressions in basically two ways:

> Any finite sequence of N expressions (think of starting with terms) may be connected by means of N–1 blanks to form a new expression.
>
> Any finite sequence of N expressions may be connected by means of an operator (, , , ...) with N–1 commas, putting one expression into each of the N slots between commas, to form a new expression.

In spirit, the System of Logical Graphs does not specify an interpretation for its connectives, i.e., does not assign them a fixed meaning in relation to logical notions like conjunction (*and*), disjunction (*or*), and so on, the idea being to find laws of logic that are invariant over variations in both operands and operators. In practice, however, it is convenient to choose either one of two common interpretations for the connectives, corresponding to what Peirce called the *Existential* and the *Entitative* versions of Logical Graphs.

We will immediately specialize to the Existential interpretation, which Peirce himself eventually came to favor, but mention this for those readers of Laws of Form who will find the Entitative alternative emphasized there. In the interpretation adopted here, we assign to the blank connective, or concatenation, the meaning of conjunction. This forces a number of other interpretive choices:

The blank term " " by itself has the value of true.

The operator (, , , ...) says "just one false" of its contents, that is to say, the expression (E1, E2, E3) has the value true just in case exactly one of the expressions E1, E2, E3 is false.

Table 2. The Existential Interpretation

Expression	Interpretation
` '	True.
()	False.
A	A.
(A)	Not A.
A B C	A and B and C.
((A)(B)(C))	A or B or C.
(A (B))	A implies B. If A then B.
(A , B)	A not equal to B. A exclusive-or B.
((A , B))	A equals B.
(A , B , C)	Just one of A, B, C is false.
((A),(B),(C))	Partition into A, B, C. Just one of A, B, C is true.
(X ,(A),(B),(C))	Partition X into A, B, C. Genus X of species A, B, C.

From these first attachments is developed the whole array of meanings for expressions, a sample of which is displayed in Table 2. The unfamiliar compound connectives that appear in the Figure will be explained later on, in the discussion of examples. In what follows, the general form of expression founded on Logical Graphs will be referred to as LG. The Existential interpretation of these expressions will be indicated as Ex.

Example 1: a Polymorphous Set **Files: poly.***

We start with an example simple enough to compare the representations by Venn diagrams, Truth tables, and our version of the syntax for propositional calculus all in a relatively short space. To enliven the exercise, we borrow an example from a book with several independent dimensions of interest, Topobiology by Gerald M. Edelman [Ede]. We find discussed there the notion of a *polymorphous set*. This is defined relative to a universe of discourse whose elements can be characterized by a number N of features. In such a universe, a polymorphous set is one that can be characterized in terms of the sets whose elements have exactly M of the N features.

The example Edelman gives [Ede, 194, Fig. 10.5] concerns sets of stimulus patterns described in terms of the features *Round*, *Doubly outlined*, and *Centrally dark*. The Target concept is one expressed by the polymorphous set "at least two of Round, Doubly outlined, or Centrally dark". Taking the symbols A = Round, B = Doubly outlined, C = Centrally dark, and using them to label the circles of a Venn diagram, we get a picture of the Target set T as the shaded region in Figure 3.

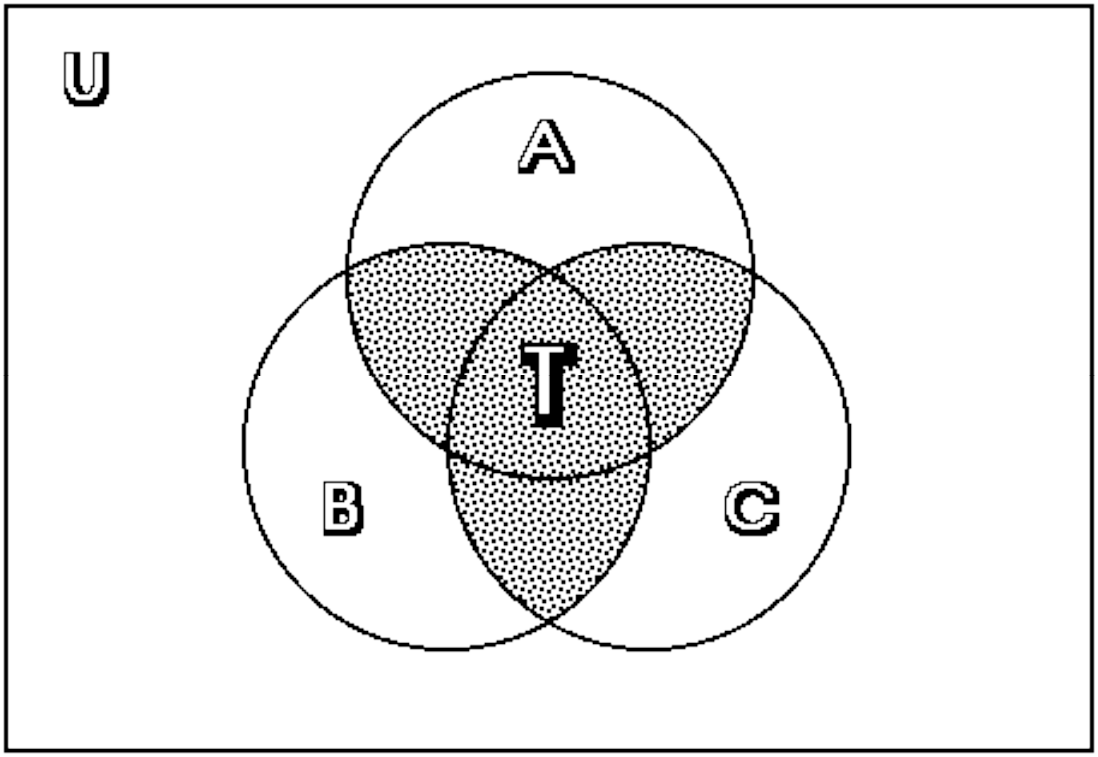


Figure 3. A Polymorphous Set

Using these symbols as sentence letters of a Truth table, let T′ mean the same thing as the expression

{ A and B } or { B and C } or { C and A }.

In other words, T′ is a truth function of the three logical variables A, B, C, and may be evaluated according to the scheme given in Table 4. In this representation the polymorphous set T appears in the guise of what some people call the *pre-image* or the *fiber* of truth under the function T′. More precisely, the three-tuples for which T′ evaluates to true are in an obvious correspondence with the shaded cells of the Venn diagram. No matter how we get down to the level of actual information, it's all pretty much the same stuff.

Table 4. Truth Table of a Polymorphous Function

A	B	C	A & B	B & C	C & A	T′
1	1	1	1	1	1	1
1	1	0	1	0	0	1
1	0	1	0	0	1	1
1	0	0	0	0	0	0
0	1	1	0	1	0	1
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

With the pictures of the Venn diagram and the Truth table before us, we have come to the verge of sensing how the word *model* is used in logic, namely as something that satisfies a description. In the Venn diagram, to be a thing of some description is to be a point x of some region Y in the universe of discourse. In the Truth table, to be a model of a proposition is to be a data vector x (an n-tuple row of the table) on which a function Y′ evaluates to true.

This makes sense to those who consider the meaning of a sentence letter to be not the sentence but its truth value instead. In this view, we may say that any data vector of this type (an n-tuple of truth values) is an *interpretation* of the proposition with n variables. An interpretation that yields a value of true is then called a *model*. For the most threadbare kind of logical system residing in Propositional Calculus this notion of model is almost too simple to deserve the name, yet it can be of service to fashion some form of continuity between the simple and the complex.

Table 5. Interpretations of a Polymorphous Function

			T′
A	B	C	1
A	B	(C)	1
A	(B)	C	1
A	(B)	(C)	0
(A)	B	C	1
(A)	B	(C)	0
(A)	(B)	C	0
(A)	(B)	(C)	0

With these preliminaries we are ready to see how the form of expression and means of evaluation used in our version of Ex (Existential graphs) may be related to the usual pictures. Looking at the initial columns of the Truth table, and having seen that the relation between the heading and the rows is one of interpretation, we can let the rows suffice to interpret themselves by replacing their 1 and 0 values with their "A" and "not A" positions regarding each proposition A in the column heads. In our current example, this gives the arrangement in Table 5. Here we have used the one-slot operator to express "not A" by (A). Also, since the parser we use in Study will later require it, we have adopted the habit of terminating each logical term (even those of one symbol) with a blank space.

(Note for later use: The parser for log files will accept <space> and also <end of line> or <carriage return> characters to terminate a word, but not <end of file> or <control-z> characters. Extra spaces and lines, not within a word, may be added freely.)

In our next transformation of the Truth table we take the rows in their present order as though they were the word sequences in some language, treating strings of the form A and (A) as distinct words, and then we unify common initial segments of these Strands to form a tree. As the leaves of this tree we take the truth functional values of the proposition, here signifying true by "*" and false by "–". This gives us the tree form shown in Figure 6.

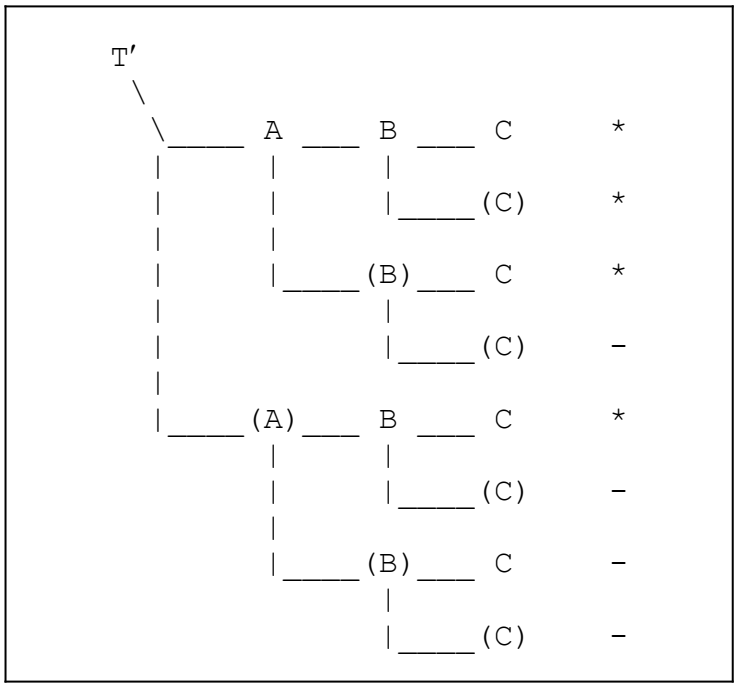


Figure 6. Tree Form of a Polymorphous Function

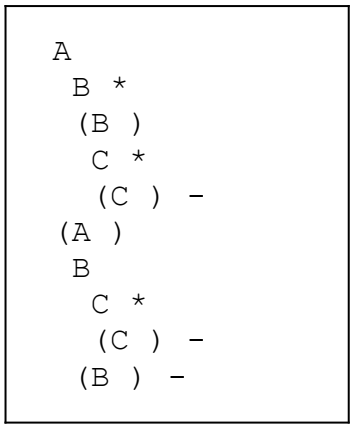


Figure 7. Outline Form of a Polymorphous Function

Finally we compress this tree into an outline form, as shown in Figure 7. This is the actual output of the Model function in the Study section when put to work on a log file containing the proposition

```
(( A B )( B C )( C A ))
```

that is,

{ A and B } or { B and C } or { C and A }

The process of making up log files will be taken up shortly, but first we need to discuss some unexplained aspects of the outline form just given. You have probably noticed that the outline produced for our current example did not cover all branches of the preceding tree in equal detail or list all its possible paths. Namely, it summarizes only the following:

```
 A     B          *
 A    (B )   C    *
 A    (B )  (C )  -
(A )   B     C    *
(A )   B    (C )  -
(A )  (B )       -
```

Thus, we can see that the tree actualized by Model has only six leaves in comparison to the eight that were possible. This is because the Model function has taken each path only as far as necessary to be certain of its value under the propositional truth function given. In this case, the first and last paths did not have to be taken through their final branch because their values were already detectable.

This is exactly the kind of advantage in efficiency we want to make the maximum possible use of, since the work of exploring 2^N rows of a table or paths of a tree becomes quickly prohibitive. But there are limits to how well we can do. Some expressions are just so complex, and there is a minimum complexity associated with all equivalent ways of expressing them. For many of the natural examples however, the universes of discourse we actually find ourselves needing to model, things are usually not so bad.

In these situations we can treat Propositional Calculus as a very simple type of declarative programming language, not too rich in what it can express but useful at any rate for certain jobs, with the Model function acting as the evaluator or interpreter of the language. Seen in this light it is not so surprising that there can be arbitrarily inefficient programs for the same task, the question is whether we can find better ones. For many of the problems that arise spontaneously, and this can only be an empirically defined class for now, it often happens that more efficient descriptions can be found by literally adding more variables, so long as these are used to supply more constraining information on the space of models.

There is one other way that we can improve the procedure for evaluating interpretations, i.e., for finding the value of a proposition at the end of a table row, tree path, or sequence of indented topic headings in an outline. In the original Truth table, each row must list the variables in the same order, but now that we have given each interpretation its own head we can let them ride off in all directions at once. In other words, once two roads have diverged they need not visit the remaining places in the same order, not when some independent advantage can be seen to doing otherwise. These remarks will become clearer when we see some larger examples where the Model function has room to exploit this strategy.

Finally, before we leave our polymorphous example, there is one last aspect of the Model function that we need to discuss, the subject of Normal Forms. We have seen the concept of a logical model appear in several guises: the true interpretations, the rows of a table or the paths of a tree that lead to a true value or *satisfy* a proposition, the cells of an indicated region in a Venn diagram.

Notice that we can use the set of models of a proposition to obtain a logically equivalent proposition whose meaning across cases is manifestly clear, being expressed by independent pieces. We do this by transforming each model into the conjunction (*and*) of its features and then forming the disjunction (*or*) of the whole set of such conjuncts. The result is called a Disjunctive Normal Form (DNF). Thus, in our example we get:

```
((    A     B
)(    A    (B )   C
)(   (A )   B     C
))
```

which means:

```
either {  A and  B  }
  or   {  A and -B and C }
  or   { -A and  B and C }
<end>
```

Here we illustrate how the log file parser lets us arrange the disjuncts on separate lines, and use extra spaces for ease of reading. The DNF that Model gave us here is probably not too impressive, since the original expression was already in such a form, and a shorter one at that, but it does the job.

There are a number of other functions in the Study section, all of which start with the normal form produced by Model and derive various abstracts of it which, depending on the proposition, may be complete enough and more clear. Rather than think up new qualifiers (like *pseudo-*, *quasi-*, *semi-*) for these new forms, we will simply refer to all of them as *normal* forms. With this we arrive at our ultimate description of the type of logical modeling process implemented here. In sum, Study is a set of functions for computing normal forms of propositions. Their essential character is that of clarifying expression while preserving meaning. The Existential graph syntax and its possibilities should become clearer with a few more examples, to which we now proceed.

Example 2: Distinction, Equation, Implication **Files: dei.***

In review of what has gone before, our version of Propositional Calculus is based on one unmarked operation of concatenation, interpreted as logical conjunction, and one N-fold operator (, , , ...) for each N > 0, which is interpreted to be true if and only if exactly one of its operands is false. When necessary, we use the notation "(, , , ...) ad" to distinguish the "just one false" operator from other functions under discussion, and we call the N-adic operators monads, dyads, triads, and so on.

Example 1 covered the use of conjunction "A B C ..." and negation "(A)", which were used to express disjunction in the form "((A)(B)(C) ...)".

Example 2 will illustrate the use of some dyadic forms for expressing:

Distinction " A ≠ B "

(logical inequality, as (A , B)

exclusive disjunction)

Equality " A = B "

(logical equivalence, as ((A , B))

if and only if)

and also, the use of conjunction and negation to express:

Implication " A => B "

(*if-then*) as (A (B)) .

On disk is a log file named “dei.log” which contains the following text:

```
( male  , female )
(( boy  , male child ))
(( girl , female child ))
( child ( human ))
```

which means:

```
male  ≠  female
boy   =  male child
girl  =  female child
child => human
```

This is a actually a single proposition, a conjunction of four lines: one distinction, two equations, and one implication. Together, these amount to a set of definitions conjointly constraining the logical compatibility of the six feature names that appear. They may be thought of as sculpting out a space of models which is some subset of the $2^6 = 64$ possible interpretations, and thereby shaping some universe of discourse.

Once this backdrop is defined, by loading the files “dei.lex” and “dei.log” at the appropriate prompts of Study, the Query function in that section allows us to conjoin additional propositions in further constraint of the underlying set of models. This has many uses, as we shall see.

To see how this example works out, you would perform the following steps:

(A) Go into the Study section of Theme 1. This you would do as follows:

(1) At the DOS prompt:	Enter “Theme1”.
	(Do not type the quotes.)
(2) At the Order prompt:	Press <space>.
	(or <comma>, or <enter>.)
(3) At *Read Lex File*:	Press <space>, then Press <enter>.
(4) At *Read Lit File*:	Press <space>, then Press <enter>.
(5) At the Index prompt:	Press <period>. (or <escape>.)
(6) At the Study prompt:	Press <space>.
	(or <comma>, or <enter>.)

(B) Load the files “dei.lex” and “dei.log” at the proper read prompts:

(1) At *Read Lex File*:	Press <space>,
	then Enter “dei.lex”.
(2) At *Read Log File*:	Press <space>,

then Enter “dei.log”.

(C) Skip over the next two utility prompts:

(1) At *Write Num File*: Press <period>.

(2) At *Show Ram Files*: Press <period>.

The preceding steps should bring you to the Query prompt. At this point you may attach further constraints or conditions to the proposition in the log file before executing the Model function. In general, this *query* can be any proposition in Ex form, provided it uses only those words which are already present in the lex file. Just remember to follow each word by a <space> or a <carriage return>, then press <control-z> when you have finished typing in the query.

Usually, the query is just a conjunction of one or more positive features that we want to *focus on* or *filter out* of the model space. On our first run through this example, we take the log file proposition *as is*, with no extra riders. This you would do as follows:

(A) Conjoin an empty query to the log file proposition. That is:

(1) At the Query prompt:	Press <space>.
	(or <comma>, or <enter>.)
(2) Await the message:	"< enter lines of inquiry >".
(3) Then:	Press <control-z>.

(B) Skip over the next utility prompt, and run the Model function, thus:

(1) At *Show Files*:	Press <period>.
(2) At the Model prompt:	Press <space>.

Model displays a running tab of how much free memory space it has left. On some of the harder problems you may think of to give it, Model may run out of free memory and terminate, abnormally exiting Theme 1. Sometimes it helps to:

(1) rephrase the problem in logically equivalent but rhetorically more felicitous ways, or

(2) think of additional facts that are taken for granted but not expressed and which cannot be logically inferred by Model.

After Model has finished, it is ready to write out its "mod" file, which you may choose to show on the screen or save to a named file. At the Write Mod File prompt, you have one of three choices:

(1) To show the file on screen:	Press <space>,
	then Press <enter>.
(2) To save the file to disk:	Press <space>,
	then Enter a file name.
(3) To skip the file write:	Press <period>.

Mod files are usually too long to see (or care to see) all at once on the screen, so it is often best to save them for later replay (e.g., with the DOS "Type" command or an ASCII editor). This example is already on the disk file named "dei.mod", and its contents are repeated below.

Model output &
"mod" file example

```
male                                1
 female -                           2
 (female )
  girl -                            3 *
  (girl )                           4
   child                            5
    boy                             6
     human *                        7 *
     (human ) -                     8
    (boy ) -                        9 *
   (child )                        10
    boy -                          11
    (boy ) *                       12
(male )                            13 *
 female                            14
  boy -
  (boy )
   child
    girl
     human *
     (human ) -
    (girl ) -
   (child )
    girl -
    (girl ) *
 (female ) -
```

Counting the stars "*" and bars "–", we can see that Model, out of the 64 possible interpretations, has actually gone through the work of making 14 evaluations, all in order to find the 4 models allowed by the definitions.

To be clear about what this output means, the starred paths indicate all of the *complete specifications* of objects in a universe of discourse, i.e., all of the consistent feature conjunctions of maximum length, as permitted by the definitions given in the log file.

To help identify these specifications, the next function and output format, called Tenor, edits the mod file to give only the true paths, or models.

Skip the last write prompt in Model, and run the Tenor function, thus:

(1) At *Write Nom File*: Press <period>.
(2) At the Tenor prompt: Press <space>.
(3) At *Write Ten File*; Press <space>.

Here is the output of Tenor, a copy of which is on the disk file "dei.ten".

Tenor output & "ten" file example

```
male
  (female )
   (girl )
    child
     boy
      human *          <1>
    (child )
     (boy ) *          <2>
 (male )
  female
   (boy )
    child
     girl
      human *          <3>
    (child )
     (girl ) *         <4>
```

As we said, this just gives a transcript of the models in the previous output. These specifications, or feature conjunctions, with positive and negative features in order of their consideration along the various paths, are as follows:

```
<1>    male   (female ) (girl )  child    boy     human
*
<2>    male   (female ) (girl ) (child ) (boy  )
*
<3>   (male )  female   (boy  )  child    girl    human
*
<4>   (male )  female   (boy  ) (child ) (girl )
*
```

Notice that Model, as reflected in this abstract, did not consider the six positive features in the same order along each path. This is because the algorithm was designed to proceed opportunistically in its attempt to reduce the original proposition by a series of decisions.

Note also that Model is something of a *lazy* evaluator, quitting work when a value is determined by less than the full set of variables. This is the reason why paths <2> and <4> are not ostensibly of the maximum length. According to this lazy mode of understanding, any path not specified on a set of features really stands for the whole bundle of paths arrived at by freely varying those features. In short, specifications <2> and <4> summarize four models, with the choice "human / not human" being left open.

The next two functions in the Study section, “Canon” and “Sense”, extract further derivatives of the normal forms produced by Model and Tenor. Both of these functions take the set of model paths and simply throw away the negative labels. You may think of these as the *rose colored glasses* or *job interview* normal forms, in that they try to say all that’s true, so long as it can be expressed in positive terms. Generally, this would mean losing a lot of information, and the result could no longer be expected to have the property of logical equivalence with the original proposition.

Fortunately, however, it seems that this type of positive projection of the whole truth is just what is possible, most needed, and most clear in many of the “natural” examples, i.e., from the domains of natural language and natural conceptual kinds. Here, where most of the features are redundantly coded, e.g., the way that adult = (child) and child = (adult), the positive paths by themselves are often sufficiently expressive.

Canon merely censors its printing of the negative labels as it traverses the model tree. This leaves the positive labels in their original columns of the outline form, giving it a slightly skewed appearance. This can be misleading unless you already know what you are looking for. However, this format is computationally quick, and often suffices.

Here is the output of Canon, a copy of which is on the disk file “dei.can”.

Canon output &
“can” file example

```
male
   child
    boy
     human
 female
   child
    girl
     human
```

Each of the normal forms has an outline text file and a generic disk file format in which it may be saved. The latter, not normally matter for human reading tastes, are prompted for with the “n” trigrams: *nom*, *net*, *nac*, *nes*, for the Model, Tenor, Canon, Sense forms, respectively. These files are not further discussed in the current documentation, but we mention them so you won’t be surprised by their strange output spilling onto the screen.

From here on, we abbreviate our discussion a bit and do not mention all of the functions, irrelevant to the moment, that might appear. We assume that you will simply skip the prompts not treated here, by pressing a <period> on their occurrence.

Sense does the extra work required to place the positive labels of the model tree at their proper level in the outline. You will see a preliminary request to “recycle memory” so that Sense can get any extra memory space it might need to do the job. Of course, in some situations there might not be enough, and Sense can fail at this point, exiting Theme 1.

Here is the output of Sense, from the disk file named “dei.sen”.

```
Sense output &
“sen” file example
```

```
male
 child
  boy
   human
female
 child
  girl
   human
```

The Canon and Sense outlines for this Example illustrate a certain kind of general circumstance which needs to be noted. Recall the model paths or feature specifications numbered <2> and <4> in the discussion of the output for Tenor. These paths reflected Model’s discovery that the Venn diagram cells for male or female non-children and non-humans were not excluded by the definitions given in the log file.

In the abstracts given by Canon and Sense, the specifications <2> and <4> have been subsumed, or absorbed unmarked, under the general topics of their respective genders, male or female. This happens because no purely positive features were supplied to distinguish the non-child and non-human cases.

After Sense is done, the Study function returns to the Query prompt, for possible further consideration of the same log file.

To exit quickly from Theme 1, just keep pressing <period> or <escape> at each prompt until you get back to DOS.

Example 3: Partition, Genus and Species **Files: gen.***

If	(A1 , A2 , A3 , ...)	means that just one of	
		A1 , A2 , A3 , ...	is false,
then	((A1),(A2),(A3), ...)	must mean that just one of	
		(A1),(A2),(A3), ...	is false,
in other words, that just one of		A1 , A2 , A3 , ...	is true.

Thus we have an efficient means to express and enforce a partition of the space of models, the condition that a number of features or propositions are to be held in mutually exclusive and exhaustive disjunction. This supplies a much needed bridge between the binary domain of two values and any other domain with a finite number of feature values.

Another variation on this theme allows us to maintain the subsumption of many separate *species* under an explicit *genus*. To explain, let us view the form of expression

(A , (A1) , (A2) , (A3))

and consider what it would mean for this to be true. We see two cases:

If A is true, then just one of the expressions (A1), (A2), (A3) must be false, and so just one of the expressions A1, A2, A3 must be true.

If A is false, then each one of the expressions (A1), (A2), (A3) must be true, and so each one of the expressions A1, A2, A3 must be false.

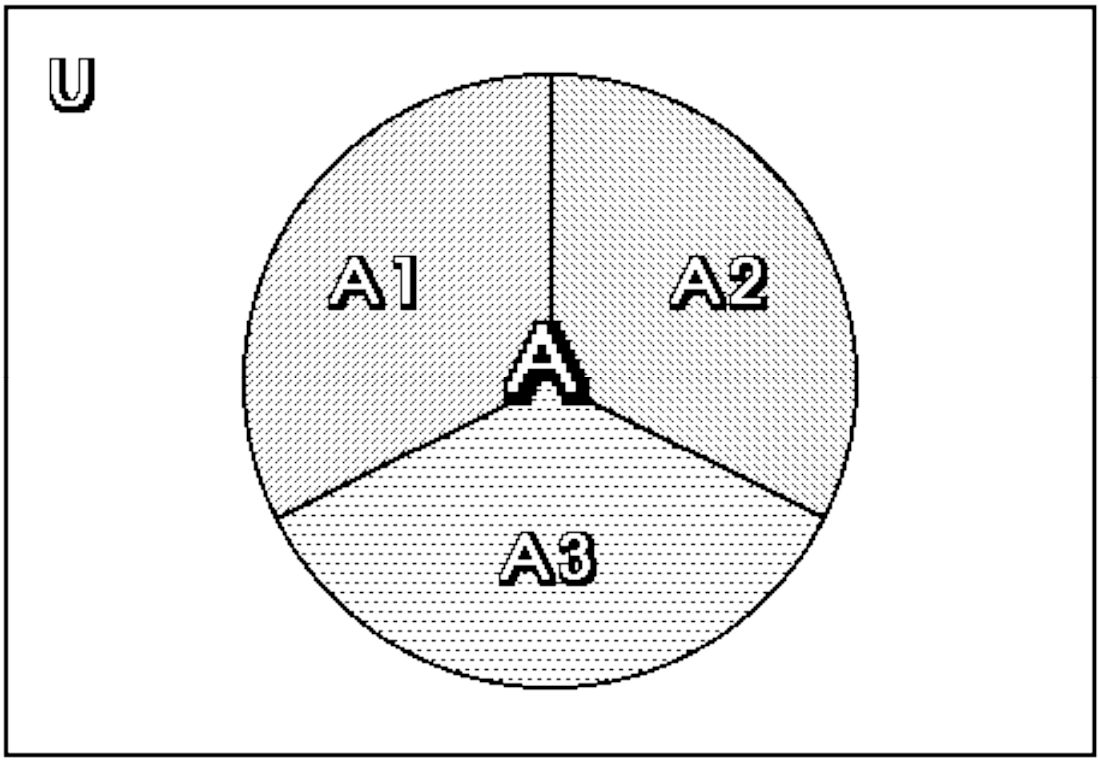


Figure 8. Genus and Species

The Venn diagram in Figure 8 illustrates this situation. In Example 3, we look at how these forms of partition and subsumption can interact in structuring a space of feature specifications.

In this Example, we describe a universe of discourse in terms of five features: *living_thing*, *non_living*, *animal*, *vegetable*, *mineral*; and we construe it as being subject to four rules:

(1) Everything is either a living_thing or non_living, but not both.
(2) Everything is either animal, vegetable, or mineral, but no two of these together.
(3) A living_thing is either animal or vegetable, but not both,
and everything animal or vegetable is a living_thing.
(4) Everything mineral is non_living.

(We have illustrated in addition here how the log file parser accepts the underscore character "_" , sometimes read "bound to", as any other character in its alphabet. This allows binding several of our *words* into one *term* for the parser.)

These notions and constructions are expressed in the log file shown below:

Logical input
File: gen.log

```
( living_thing , non_living )
(( animal ),( vegetable ),( mineral ))
( living_thing ,( animal ),( vegetable ))
( mineral ( non_living ))
```

which means:

```
     living_thing  ≠  non_living
par: { universe } -> { animal, vegetable, mineral }
par: living_thing -> { animal, vegetable }
     mineral => non_living
```

Here, “par: { universe } -> { X, Y, Z }” is short for an assertion that the universe is partitioned into sets corresponding to the features X, Y, Z;

Also, “par: A -> { B, C }” asserts that “A is partitioned into B and C”.

It is probably enough just to list the outputs of Model, Tenor, and Sense when run on the preceding log file. Using the same format and labeling as before, we may note that Model has, from $2^5 = 32$ possible interpretations, made 11 evaluations, and found 3 models answering the general descriptions.

Model outline
File: gen.mod

```
living_thing
 non_living -          1
 (non_living )
  mineral -            2
  (mineral )
   animal
    vegetable -        3
    (vegetable ) *     4 *
   (animal )
    vegetable *        5 *
    (vegetable ) -     6
(living_thing )
 non_living
  animal -             7
  (animal )
   vegetable -         8
   (vegetable )
```

```
     mineral *          9 *
     (mineral ) -      10
  (non_living ) -      11
```

Tenor outline
File: gen.ten

```
 living_thing
  (non_living )
   (mineral )
    animal
     (vegetable ) *     <1>
    (animal )
     vegetable *        <2>
 (living_thing )
  non_living
   (animal )
    (vegetable )
     mineral *          <3>
```

Sense outline
File: gen.sen

```
  living_thing
   animal
   vegetable
  non_living
   mineral
```

Example 4: Molly's World **Files: molly.***

Example 4 is taken from the literature on computational learning theory, adapted with some changes from the example called "Molly's Problem" in the paper *Learning with Hints* by Dana Angluin [Ang]. We quote Angluin's motivational description to set up the problem:

> Imagine that you have become acquainted with an alien named Molly from the planet Ornot, who is currently employed in a day-care center. She is quite good at propositional logic, but a bit weak on knowledge of Earth. So you decide to formulate the beginnings of a propositional theory to help her label things in her immediate environment.

The purpose of this quaint pretext is, of course, to make sure the reader appreciates the constraints of the problem: that no extra savvy is fair, all facts must be presumed or deduced upon the immediate premises.

Our use of this example is not directly relevant to the purposes of the discussion from which it is borrowed, so we simply give our version of it without comment on those issues.

Here is our rendition of the initial knowledge base delimiting Molly's World:

File: molly.log

```
( object ,( toy ),( vehicle ))
(( small_size ),( medium_size ),( large_size ))
(( two_wheels ),( three_wheels ),( four_wheels ))
(( no_seat ),( one_seat ),( few_seats ),( many_seats ))
( object ,( scooter ),( bike ),( trike ),( car ),( bus ),( wagon ))
( two_wheels    no_seat            ,( scooter ))
( two_wheels    one_seat    pedals ,( bike ))
( three_wheels  one_seat    pedals ,( trike ))
( four_wheels   few_seats   doors  ,( car ))
( four_wheels   many_seats  doors  ,( bus ))
( four_wheels   no_seat     handle ,( wagon ))
( scooter           ( toy  small_size ))
( wagon             ( toy  small_size ))
( trike             ( toy  small_size ))
( bike  small_size  ( toy ))
( bike  medium_size ( vehicle ))
( bike  large_size  )
( car               ( vehicle  large_size ))
( bus               ( vehicle  large_size ))
( toy               ( object ))
( vehicle           ( object ))
```

All the forms used in the preceding log file will probably be familiar from earlier discussions. The purpose of one or two constructions may, however, be a little obscure.

The rule "(bike large_size)" , for example, merely says that nothing can be both a bike and large_size.

The rule "(three_wheels one_seat pedals , (trike))" says that anything with all the features of three_wheels, one_seat, and pedals is excluded from being anything but a trike. In short, anything with just those three features is equivalent to a trike. Recall that the form "(A , B)" may be interpreted to assert either the exclusive disjunction or the inequality of A and B.

The rules have been stated in this particular way simply to imitate the style of rules in the reference example.

This last point brings up an important issue, the question of *rhetorical* differences in expression and their potential impact on the *pragmatics* of computation. Unfortunately, we must abbreviate our discussion of this topic for now, and only mention the following facts.

Logically equivalent expressions, even though they must lead to logically equivalent normal forms, may have very different characteristics with regard to efficiency of processing.

Thus, all four of the forms

"((A , B))" "(A , (B))" "((A) , B)" "((B , A))"

are equally succinct ways of saying that A is logically equivalent to B, yet each can have different effects on the route that Model takes to arrive at its answer. Apparently, some equalities are more equal than others.

These effects occur partly because the algorithm chooses to make cases of variables on a basis of *leftmost shallowest first*, but their impact can be complicated by interactions each expression has with the context it occupies. The lesson is that it is probably best not to bother too much about these problems, but just experiment with different ways of expressing equivalent information until you get a sense of what works best in various situations.

We think you will be happy to see only the ultimate Sense of Molly's World, so here it is:

Sense outline
File: molly.sen

```
object
 two_wheels
  no_seat
   scooter
    toy
     small_size
  one_seat
   pedals
    bike
     small_size
      toy
     medium_size
      vehicle
 three_wheels
  one_seat
   pedals
    trike
     toy
      small_size
 four_wheels
  few_seats
   doors
    car
     vehicle
      large_size
  many_seats
   doors
    bus
     vehicle
      large_size
  no_seat
   handle
    wagon
     toy
      small_size
```

This outline is not the Sense of the unconstrained log file, but a result of running Model with a Query on the single feature "object". Using this focus helps to make more relevant Sense of Molly's World.

The logical paradigm from which this Example was derived is that of *propositional Horn clause* theories. These clauses are of three kinds:

```
<1>   " A and B and C and  ...  =>  Z "
<2>   " Z "
<3>   " not { A and B and C and ... } "
```

where the proposition letters A, B, C, ... , Z are restricted to what we would call single positive features, not themselves negated or otherwise complex expressions.

For comparison, in the syntax of Ex these forms would look like:

```
<1>    ( A B C ... ( Z ))
<2>      Z
<3>    ( A B C ... )
```

The style of deduction in Horn clause logics is essentially proof-theoretic in character, with the burden of proof falling on implication ("=>") and inference (*modus ponens* or *resolution*), Cf: [Llo, MaW].

In contrast, the method used here is substantially model-theoretic, the stress being to start from more general forms of expression for laying out the facts (e.g., distinctions, equations, partitions) and to work toward results that maintain logical equivalence with their origins.

What all of this has to do with the output above is this: From the perspective adopted here, almost any theory (e.g., one founded on the postulates of Molly's World) will have far more models than the implicational and inferential mode of reasoning is designed to discover. We will be forced to confront them, however, if we try to run Model on a large set of implications.

The typical Horn clause interpreter gets around this difficulty only by a stratagem that takes clauses to mean something other than what they say, that is, by distorting the semantics. Our Model, on the other hand, has no such finesse.

This explains why it was necessary to impose the prerequisite "object" constraint on the log file for Molly's world. It only supplied what we usually take for granted, in order to obtain a set of models we would normally think of as being the import of the definitions.

Example 5: Jets and Sharks **Files: jas.***

The propositional calculus based on the boundary operator can be interpreted in a way that resembles the logic of activation states and competition constraints in certain neural network models. One way to do this is by interpreting the blank or unmarked state as the resting state of a neural pool, the bound or marked state as its activated state, and by representing a mutually inhibitory pool of neurons A, B, C in the expression (A , B , C). To illustrate this possibility,

we transcribe a well-known example from the parallel distributed processing literature [McR] and work through two of the associated exercises as portrayed in Existential Graph format.

File: jas.log

```
(( art    ),( al   ),( sam  ),( clyde ),( mike  ),
 ( jim    ),( greg ),( john ),( doug  ),( lance ),
 ( george ),( pete ),( fred ),( gene  ),( ralph ),
 ( phil   ),( ike  ),( nick ),( don   ),( ned   ),( karl ),
 ( ken    ),( earl ),( rick ),( ol    ),( neal  ),( dave ))

( jets , sharks )

( jets ,         ( art    ),( al   ),( sam  ),( clyde ),( mike  ),
                 ( jim    ),( greg ),( john ),( doug  ),( lance ),
                 ( george ),( pete ),( fred ),( gene  ),( ralph ))
( sharks ,       ( phil ),( ike  ),( nick ),( don ),( ned  ),( karl ),
                 ( ken  ),( earl ),( rick ),( ol  ),( neal ),( dave ))

(( 20's ),( 30's ),( 40's ))

( 20's ,         ( sam    ),( jim  ),( greg ),( john ),( lance ),
                 ( george ),( pete ),( fred ),( gene ),( ken   ))
( 30's ,         ( al   ),( mike ),( doug ),( ralph ),( phil ),( ike  ),
                 ( nick ),( don  ),( ned  ),( rick  ),( ol   ),( neal ),
                 ( dave ))
( 40's ,         ( art ),( clyde ),( karl ),( earl ))

(( junior_high ),( high_school ),( college ))

( junior_high ,  ( art  ),( al    ),( clyde  ),( mike  ),( jim ),
                 ( john ),( lance ),( george ),( ralph ),( ike ))
( high_school ,  ( greg ),( doug ),( pete ),( fred ),( nick ),
                 ( karl ),( ken  ),( earl ),( rick ),( neal ),( dave ))
( college ,      ( sam ),( gene ),( phil ),( don ),( ned ),( ol ))

(( single ),( married ),( divorced ))

( single ,       ( art  ),( sam  ),( clyde ),( mike ),( doug ),( pete ),
                 ( fred ),( gene ),( ralph ),( ike  ),( nick ),( ken  ),
                 ( neal ))
( married ,      ( al  ),( greg ),( john ),( lance ),( phil ),
                 ( don ),( ned  ),( karl ),( earl  ),( ol   ))
( divorced ,     ( jim ),( george ),( rick ),( dave ))

(( bookie ),( burglar ),( pusher ))

( bookie ,       ( sam  ),( clyde ),( mike ),( doug ),
                 ( pete ),( ike   ),( ned  ),( karl ),( neal ))
( burglar ,      ( al     ),( jim ),( john ),( lance ),
                 ( george ),( don ),( ken  ),( earl  ),( rick ))
```

```
( pusher ,       ( art   ),( greg ),( fred ),( gene ),
                 ( ralph ),( phil ),( nick ),( ol   ),( dave ))
```

We now apply Study to the proposition defining the Jets and Sharks data base.

With a query on the name "ken" we obtain the following output, giving all the features associated with Ken:

File: ken.sen

```
ken
 sharks
  20's
   high_school
    single
     burglar
```

With a query on the two features "college" and "sharks" we obtain the following outline of all features satisfying these constraints:

File: cos.sen

```
college
 sharks
  30's
   married
    bookie
     ned
    burglar
     don
    pusher
     phil
     ol
```

From this we discover that all college Sharks are 30-something and married. Further, we have a complete listing of their names broken down by occupation, as no doubt all of them will be, eventually.

Chapter 3: Linear Topics

The Differential Theory of Qualitative Equations

To denote lists of propositions and detail their components, we use notations like

a = <a, b, c>, **p** = <p, q, r>, **x** = <x, y, z>,

or, in more complicated situations,

x = <x1, x2, x3>, **y** = <y1, y2, y3>, **z** = <z1, z2, z3>.

In a universe where some region is ruled by a proposition, it is natural to ask whether we can change the value of that proposition by changing the features of a current state.

Given a Venn diagram with a shaded region and starting from any cell in the universe, what sequences of feature changes, what traverses of cell walls, will take us from shaded to unshaded areas, or the reverse?

In order to discuss questions of this type, it is useful to define several *operators* on functions. An operator is nothing more than a function between sets that happen to have functions as members. A typical operator F takes us from thinking about a given function f to thinking about another function g. To express the fact that g can be obtained by applying F to f, we write g = Ff.

The first operator, E, associates with a function f: A –> B, another function Ef, where Ef: A x A –> B is defined by

Ef(x, y) = f(x + y).

E is called a shift operator because it takes us from contemplating the value of f at a place x to considering the value of f at a shift of y away. It tells us the absolute effect on f of changing its argument from x by an amount y.

The second operator, D, associates with a function f: A –> B another function Df, where Df: A x A –> B is defined by

Df(x, y) = Ef(x, y) – f(x),

or, equivalently,

Df(x, y) = f(x + y) – f(x).

D is called a difference operator because it gives us the relative change in the value of f along the shift from x to x + y.

In practice, one of the variables x or y is usually considered to be “less variable” than the other, being fixed in the context of a concrete discussion. Thus we may we see any one of the following idioms:

1. Df: A x A –> B
   Df(c, x) = f(c + x) – f(c)

Here, c is held constant and Df(c, x) is regarded mainly as a function of the second variable x, giving the relative change in f at various distances x from a center c.

2. Df: A x A –> B
   Df(x, h) = f(x + h) – f(x)

Here, h is either a constant (usually 1) in discrete contexts, or a variably "small" amount (near to 0) over which a limit is taken in continuous contexts. Df(x, h) is regarded mainly as a function of the first variable x, giving the differences in the value of f between x and a neighbor h away as x ranges over various locations.

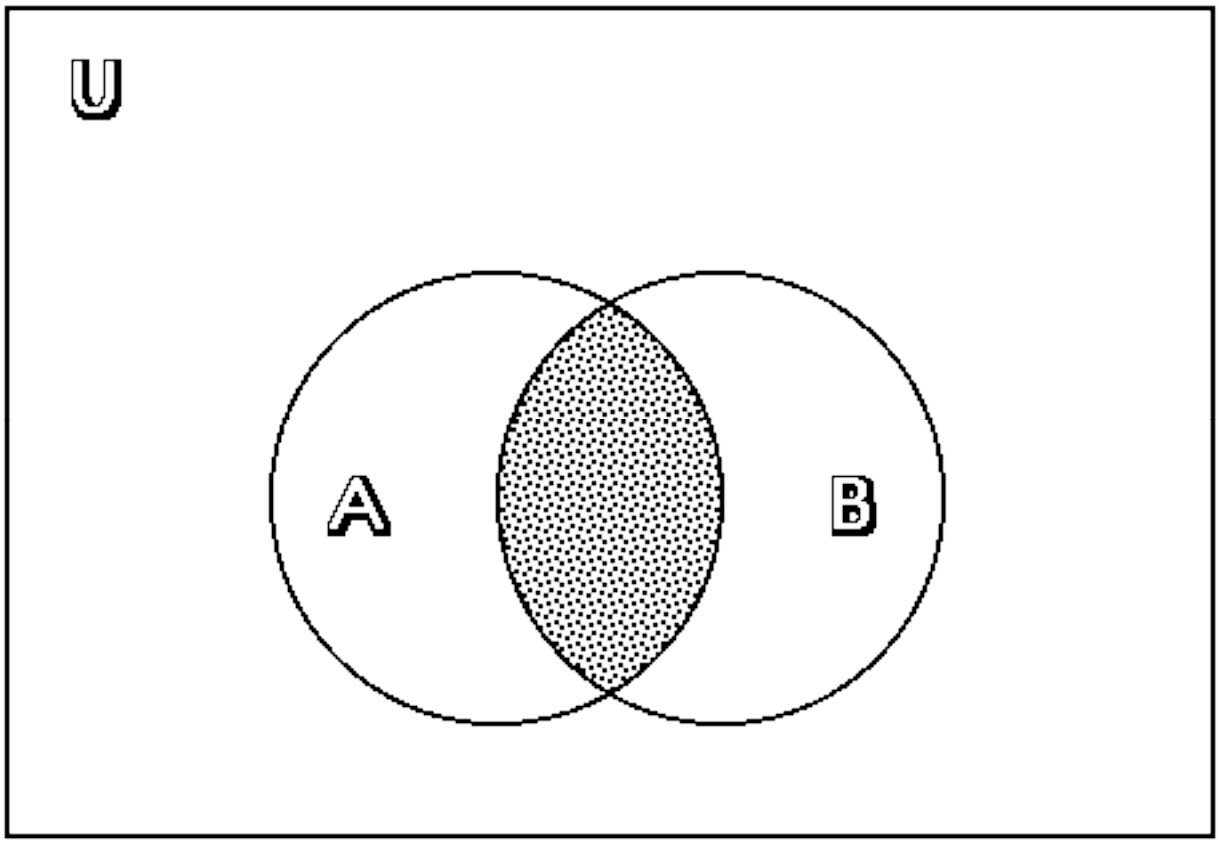
3. Df: A x A –> B
   Df(x, dx) = f(x + dx) – f(x)

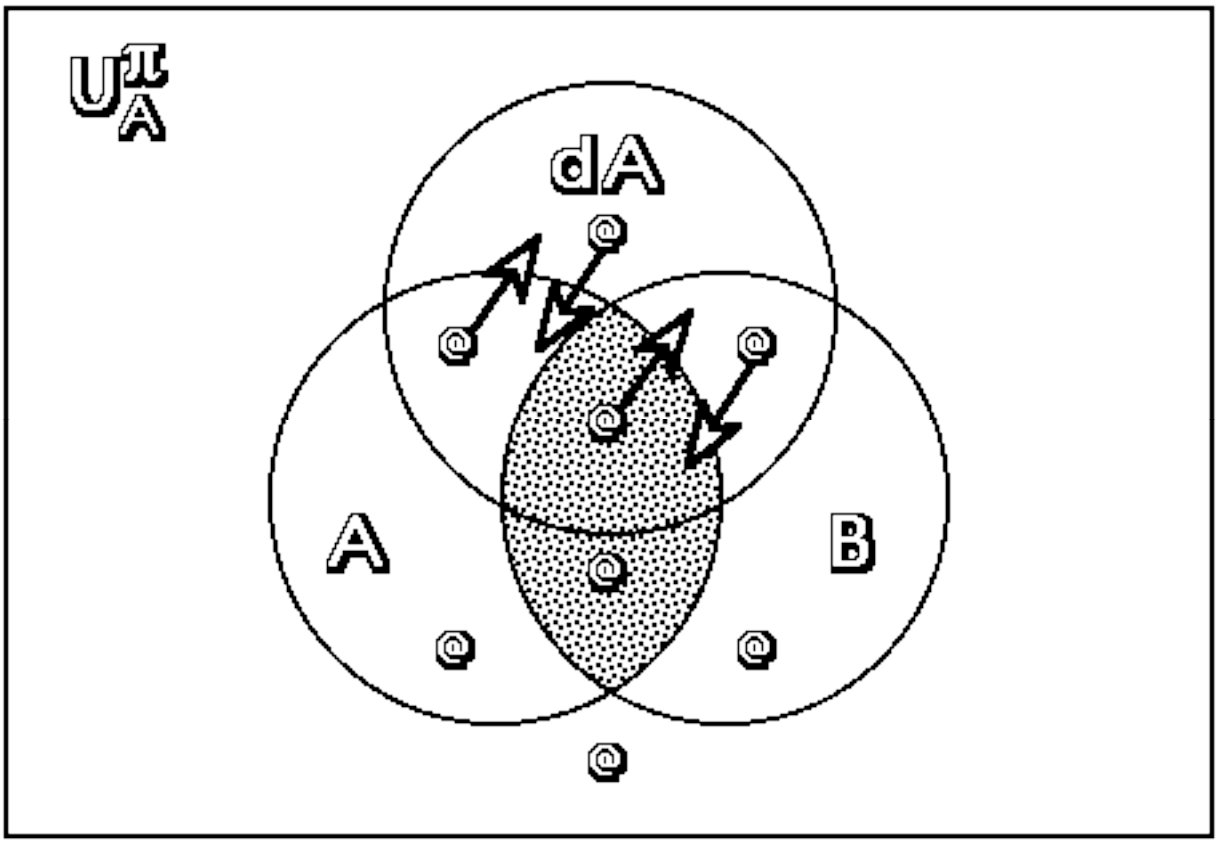
This is another variant of the previous form, with dx denoting the small changes contemplated in x.

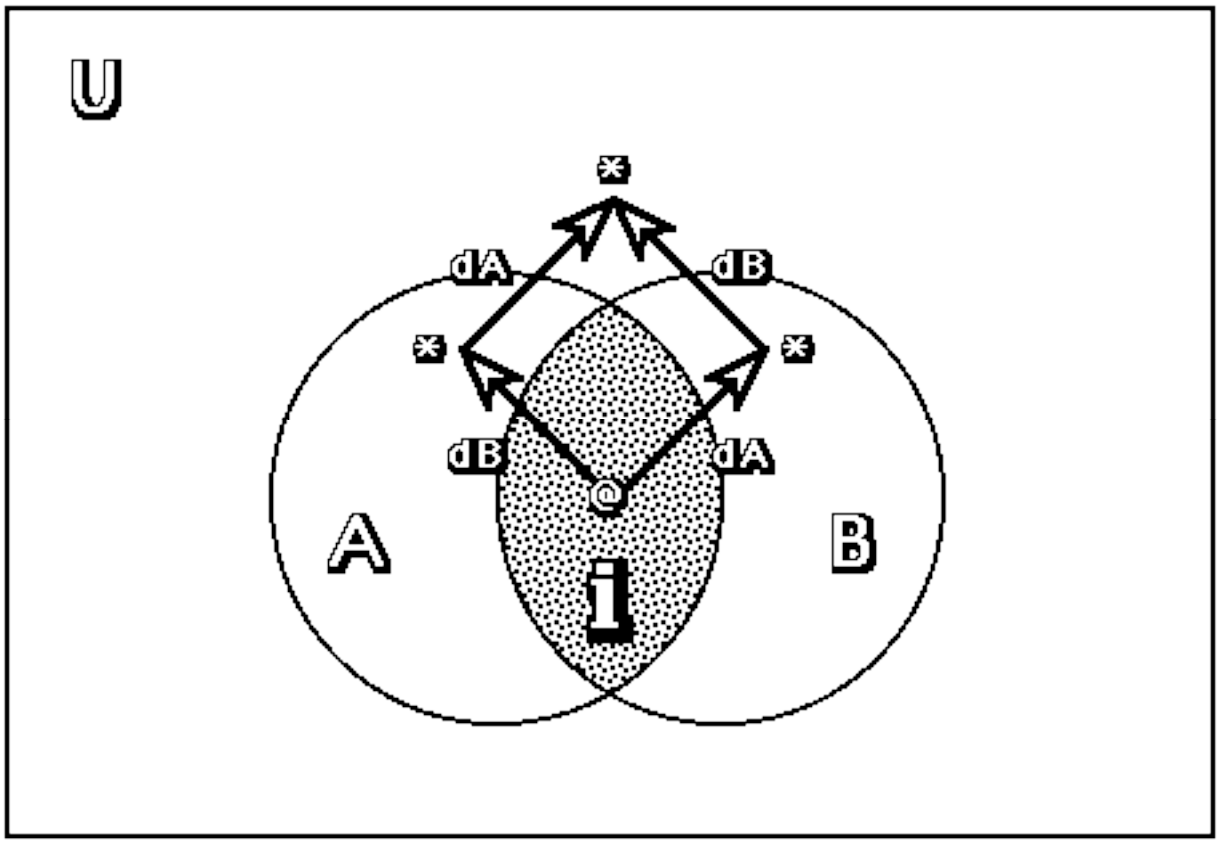
Imagine that we are sitting in one of the cells of a Venn diagram, contemplating the walls. There are N of them, one for each positive feature x1, ..., xN in our universe of discourse. Our particular cell is described by a concatenation of N signed assertions, positive or negative, regarding each of these features. Are we locked into this interpretation?

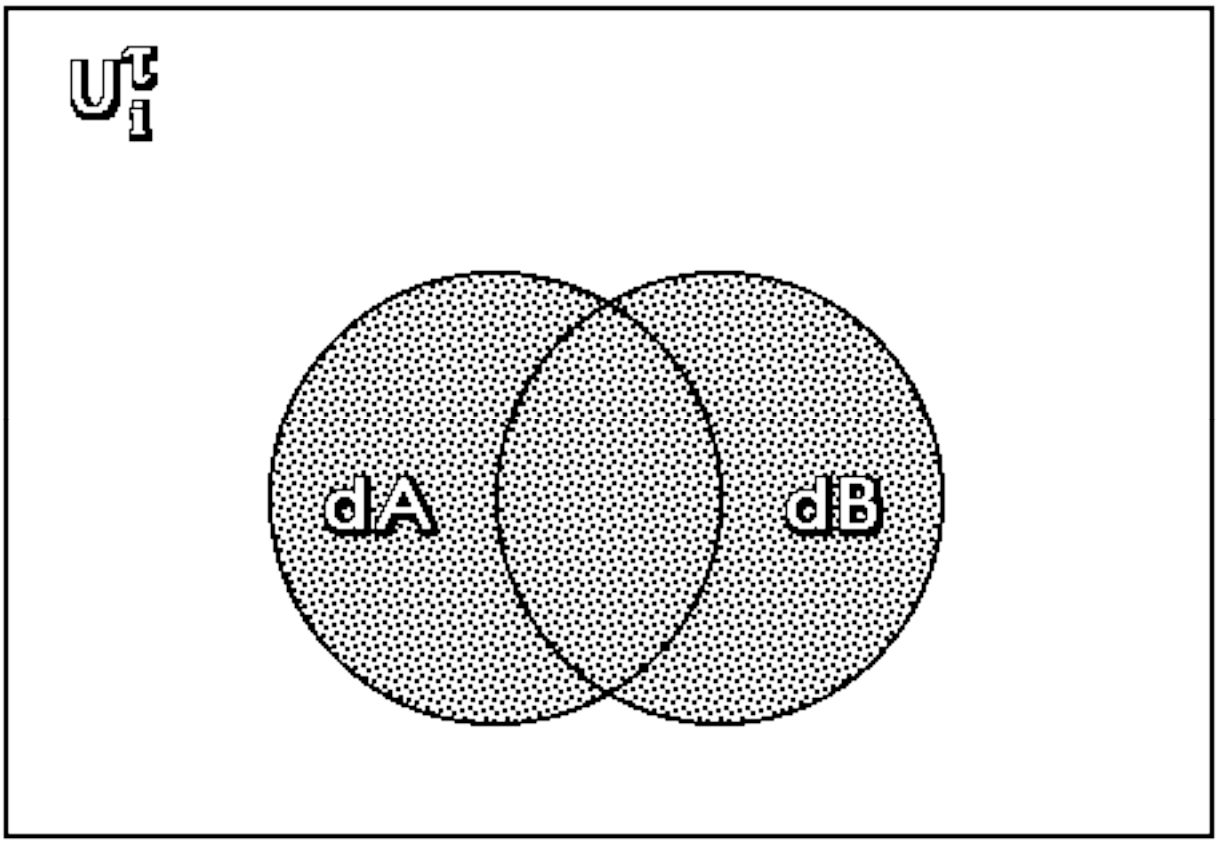
With respect to each edge x of the cell we consider a test proposition dx, to determine our decision to differ on x. If dx is true we decide to cross over edge x at some point in the future. To reckon the effect of several such decisions on the value of the reigning proposition, we transform that proposition by making the following set of substitutions everywhere in its expression:

```
Substitute ( x1 , dx1 ) for x1,
Substitute ( x2 , dx2 ) for x2,
   ... ,
Substitute ( xN , dxN ) for xN.
```


U
A
B


U_A^{π}
dA
A
B


U
*
dA
dB
*
*
dB
dA
@
A
B
i



For a more complex example, consider the polymorphous set T of Example 1 and focus on the central cell described by the features A B C. The proposition or truth function T′ describing T is:

```
(( A B )( B C )( C A ))
```

Conjoining the query that specifies the center cell yields:

```
(( A B )( B C )( C A )) A B C
```

and we know the value of the interpretation by whether this expression issues in a model.

The result of the shift transformation on the original proposition is:

```
((   ( A , dA )( B , dB )
)(   ( B , dB )( C , dC )
)(   ( C , dC )( A , dA )
))
```

Conjoining a query on the center cell yields:

```
((   ( A , dA )( B , dB )
)(   ( B , dB )( C , dC )
)(   ( C , dC )( A , dA )
))

A B C
```

and the models of this expression tell us which feature changes will take us from our current interpretation (the center cell) to a true value under the proposition.

The application of the difference operator to the original proposition, conjoined with a query on the center cell, gives:

```
(
      ((   ( A , dA )( B , dB )
      )(   ( B , dB )( C , dC )
      )(   ( C , dC )( A , dA )
      ))
,
      ((   A B
      )(   B C
      )(   C A
      ))
)

A B C
```

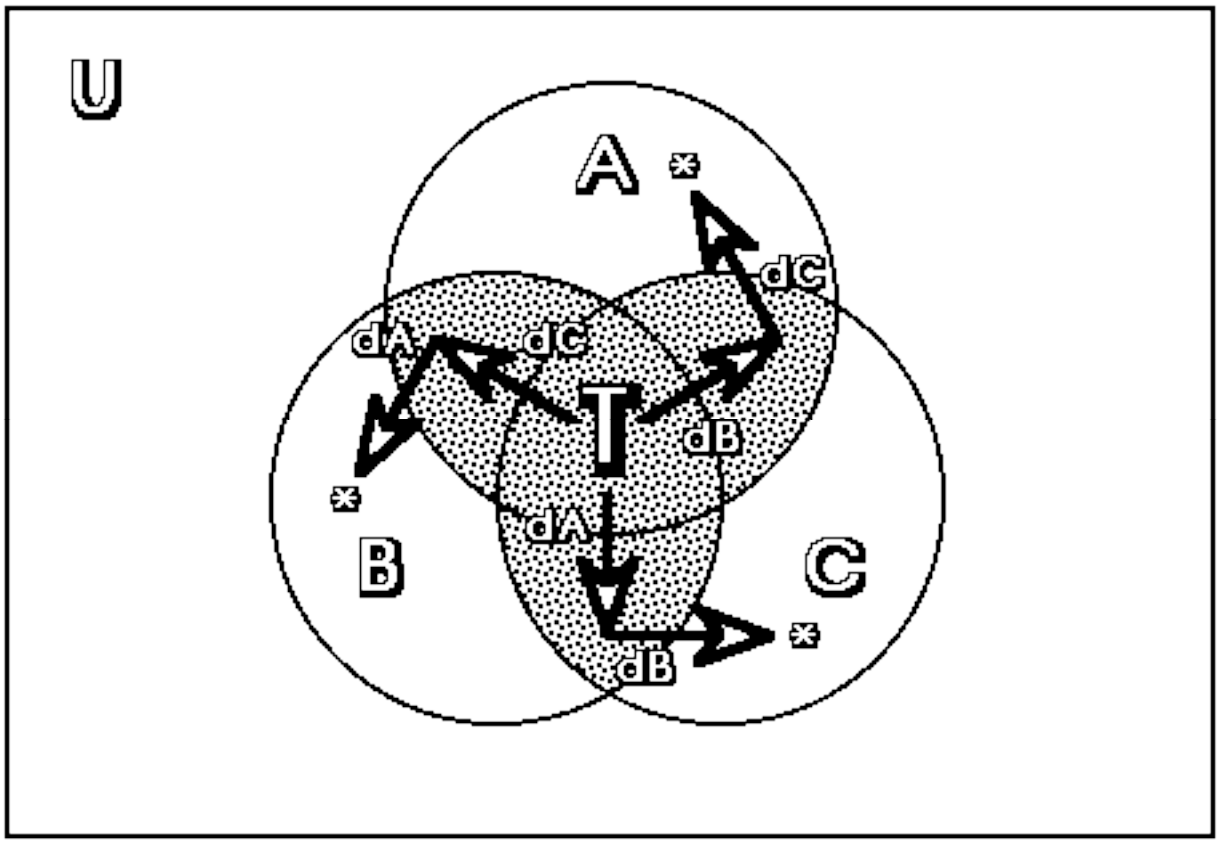
Applying Study to this last query results in the following output:

```
A
 B
  C
   dA
    dB *
    (dB )
     dC *
     (dC ) -
   (dA )
    dB
     dC *
     (dC ) -
    (dB ) -
  (C ) -
 (B ) -
(A ) -
```

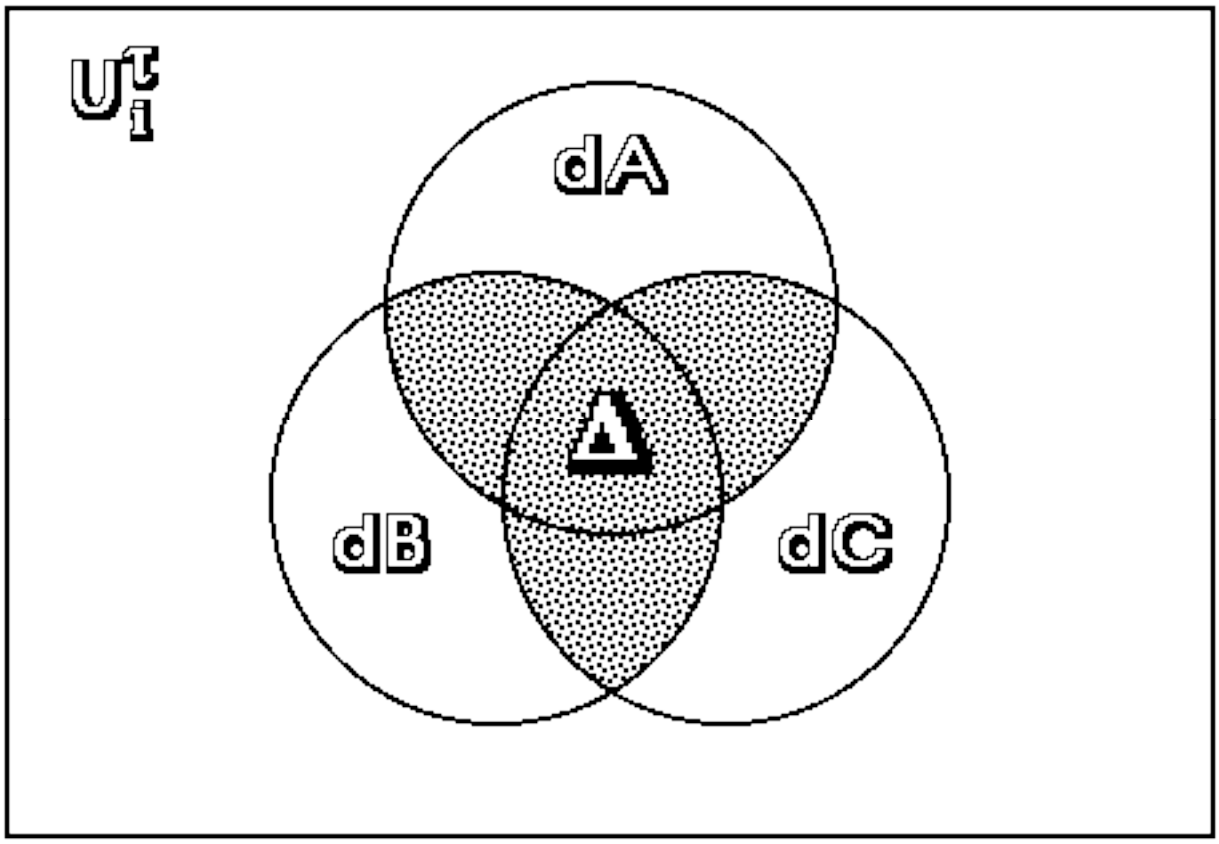
from which we pick out the following model paths:

```
<1>   A   B   C   dA   dB         *
<2>   A   B   C   dA  (dB ) dC   *
<3>   A   B   C  (dA ) dB   dC   *
```

This tells us that changing any two of the three features will take us from the center cell to outside the shaded region of the corresponding Venn diagram.



Another way of looking at this is by letting dA, dB, dC be taken as the features of another universe of discourse, called the tangent space of U with respect to the given interpretation i. Then the difference function Δ = Df(i, x) corresponds to the shaded region in the next figure.


dA
dB
dC

Chapter 4: Extra Examples

(Note: Discussion of these examples is still being written.)

1. Propositional logic example.

Disk: 1

Files: Alpha.lex + Prop.log

Ref: [Cha, 20, Example 2.12]

2. Chemical synthesis problem.

Disk: 1

Files: Chem.*

Ref: [Cha, 21, Example 2.13]

3. N Queens problem.

Disk: 2

Files: Queen*.*
Q8.*
Q5.*

Refs: [BaC, 166], [VaH, 122], [Wir, 143].

Notes: Only the 5 Queens example will run in 640K memory.
Use the "Queen.lex" file to load the "Q5.eg*" log files.

4. Five Houses problem.

Disk: 2

Files: House.*

Ref: [VaH, 132].

Notes: Will not run in 640K memory.

5. Graph coloring example.

Disk: 2

Files: Color.*

Ref: [Wil, 196].

6. Examples of Cook's Theorem in computational complexity, that propositional satisfiability is NP-complete.

Disk: 2

Files: StiltN.* = "Space and Time Limited Turing Machine", with N units of space and N units of time.

StuntN.* = "Space and Time Limited Turing Machine", for computing the parity of a bit string, with Number of Tape cells of input equal to N.

Ref: [Wil, 188-201].

Notes: Can only run Turing machine example for input of size 2. Since the last tape cell is used for an end-of-file marker, this amounts to only one significant digit of computation.

Use the "Stilt3.lex" file to load the "Stunt2.egN" files. Their Sense file outputs appear on the "Stunt2.seN" files.

7. Fabric knowledge base.

Disk: 3

Files: Fabric.*
Fab.*

Ref: [MaW, 8-16].

8. Constraint Satisfaction example.

Disk: 4

Files: Consat1.*
Consat2.*

Ref: [Win, 449, Exercise 3-9].

Notes: Attributed to Kenneth D. Forbus.

9. Linear Topics: Dungeons and Diagrams.

Disk: 4

Files:

Delta.*	Prime.*	
Polyshif.*	Efax.*	Epoly.*
Polydiff.*	Dfax.*	Dpoly.*

References

[Ang] Angluin, Dana
(1989) "Learning with Hints", in: *Proceedings of the 1988 Workshop on Computational Learning Theory*, ed: D. Haussler & L. Pitt, Morgan Kaufmann, San Mateo, CA.

[BaC] Ball, W.W. Rouse, & Coxeter, H.S.M.
(1987) *Mathematical Recreations and Essays*, 13th ed., Dover, New York, NY.

[Cha] Chang, Chin-Liang & Lee, Richard Char-Tung
(1973) *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, NY.

[DDQ] Denning, Peter J., Dennis, Jack B., and Qualitz, Joseph E.
(1978) *Machines, Languages, and Computation*, Prentice-Hall, Englewood Cliffs, NJ.

[Ede] Edelman, Gerald M.
(1988) *Topobiology: An Introduction to Molecular Embryology*, Basic Books, New York, NY.

[Llo] Lloyd, J.W.
(1984) *Foundations of Logic Programming*, Springer-Verlag, Berlin.

[MaW] Maier, David & Warren, David S.
(1988) *Computing with Logic: Logic Programming with Prolog*, Benjamin/Cummings, Menlo Park, CA.

[McR] McClelland, James L. and Rumelhart, David E.
(1988) *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA.

[P1] Peirce, Charles Sanders
(1931-1960) *Collected Papers of Charles Sanders Peirce*, ed: Charles Hartshorne, Paul Weiss, & Arthur W. Burks, Harvard University Press, Cambridge, MA.

[P2] (1976) *The New Elements of Mathematics*, ed: Carolyn Eisele, Mouton, The Hague.

[P3] (1966) *Charles S. Peirce: Selected Writings; Values in a Universe of Chance*, ed: Philip P. Wiener, Dover, New York, NY.

[SpB] Spencer Brown, George
(1969) *Laws of Form*, George Allen & Unwin, London, UK.

[VaH] Van Hentenryck, Pascal
(1989) *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA.

[Wil] Wilf, Herbert S.
(1986) *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

[Win] Winston, Patrick Henry
(1984) *Artificial Intelligence*, 2nd ed., Addison-Wesley, Reading, MA.

[Wir] Wirth, Niklaus
(1976) *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ.

Appendix A: Sources

Aristotle: On Interpretation

Chapter 1

{1} Words spoken are symbols or signs of affections or impressions of the soul; written words are the signs of words spoken. As writing, so also is speech not the same for all races of men. But the mental affections themselves, of which these words are primarily signs, are the same for the whole of mankind, as are also the objects of which those affections are representations or likenesses, images, copies.

Aristotle: Prior Analytics

Book 1 **Chapter 4**

{1} When three terms are so related to one another that the last is wholly contained in the middle and the middle is wholly contained in or excluded from the first, the extremes must admit of perfect syllogism. By 'middle term' I mean that which both is contained in another and contains another in itself, and which is the middle by its position also; and by 'extremes' (*a*) that which is contained in another, and (*b*) that in which another is contained. For if A is predicated of all B, and B of all C, A must necessarily be predicated of all C. ... I call this kind of figure the First.

Chapter 5

{2} When the same term applies to all of one subject and to none of the other, or to all or none of both, I call this kind of figure the Second; and in it by the middle term I mean that which is predicated of both subjects; by the extreme terms, the subjects of which the middle is predicated; by the major term, that which comes next to the middle; and by the minor that which is more distant from it. The middle is placed outside the extreme terms, and is first by position.

Chapter 6

{3} If one of the terms applies to all and the other to none of the same subject, or if both terms apply to all or none of it, I call this kind of figure the Third; and in it by the middle I mean that of which both the predications are made; by extremes the predicates; by the major term that which is [further from?] the middle; and by the minor that which is nearer to it. The middle is placed outside the extremes, and is last by position.

Book 2 **Chapter 21**

{1} Similarly too with the theory in the *Meno* that learning is recollection. For in no case do we find that we have previous knowledge of the individual, but we do find that in the process of induction we acquire knowledge of particular things just as though we could remember them; for there are some things which we know immediately: *e.g.*, if we know that X is a triangle we know that the sum of its angles is equal to two right angles. Similarly too in all other cases.

{2} Thus whereas we observe particular things by universal knowledge, we do not know them by the knowledge peculiar to them. Hence it is possible to be mistaken about them, not because we have contrary knowledge about them, but because, although we have universal knowledge of them, we are mistaken in our particular knowledge.

Book 2 **Chapter 23**

{1} Induction επαγωγη, or inductive reasoning, consists in establishing a relation between one extreme term and the middle term by means of the other extreme; *e.g.*, if B is the middle term of A and C, in proving by means of C that A applies to B; for this is how we effect inductions. *E.g.*, let A stand for 'long-lived', B for 'that which has no bile' and C for the long-lived individuals such as man and horse and mule. Then A applies to the whole of C, for every bileless animal is long-lived. But B, 'not having bile', also applies to all C. Then if C is convertible with B, *i.e.*, if the middle term is not wider in extension, A must apply to B.

{2} This kind of syllogism is concerned with the first or immediate premiss. Where there is a middle term, the syllogism proceeds by means of the middle; where there is not, it proceeds by induction. There is a sense in which induction is opposed to syllogism, for the latter shows by the middle term that the major extreme applies to the third, while the former shows by means of the third that the major extreme applies to the middle. Thus by nature the syllogism by means of the middle is prior and more knowable; but syllogism by induction is more apparent to us.

Book 2 **Chapter 24**

{1} We have an Example παραδειγμα when the major extreme is shown to be applicable to the middle term by means of a term similar to the third. It must be known both that the middle applies to the third term and that the first applies to the term similar to the third. *E.g.*, let A be 'bad', B 'to make war on neighbors', C 'Athens against Thebes' and D 'Thebes against Phocis'. Then if we require to prove that war against Thebes is bad, we must be satisfied that war against neighbors is bad. Evidence of this can be drawn from similar examples, *e.g.*, that war by Thebes against Phocis is bad. Then since war against neighbors is bad, and war against Thebes is against neighbors, it is evident that war against Thebes is bad. Now it is evident that B applies to C and D (for they are both examples of making war on neighbors), and A to D (since the war against Phocis did Thebes no good); but that A applies to B will be proved by means of D. ...

{2} Thus it is evident that an example represents the relation, not of part to whole or of whole to part, but of one part to another, where both are subordinate to the same general term, and one of them is known. It differs from induction in that the latter, as we saw, shows from an examination of all the individual cases that the [major] extreme applies to the middle, and does not connect the conclusion with the [minor] extreme; whereas the example does connect it and does not use all the individual cases for its proof.

Book 2 **Chapter 25**

{1} We have Reduction απαγωγη (*a*) when it is obvious that the first term applies to the middle, but that the middle applies to the last term is not obvious, yet nevertheless is more probable or not less probable than the conclusion; or (*b*) if there are not many intermediate terms between the last and the middle; for in all such cases the effect is to bring us nearer to knowledge.

{2} (*a*) *E.g.*, let A stand for 'that which can be taught', B for 'knowledge' and C for 'morality'. Then that knowledge can be taught is evident; but whether virtue is knowledge is not clear. Then if BC is not less probable or is more probable than AC, we have reduction; for we are nearer to knowledge for having introduced an additional term, whereas before we had no knowledge that AC is true.

{3} (*b*) Or again we have reduction if there are not many intermediate terms between B and C; for in this case too we are brought nearer to knowledge. *E.g.*, suppose that D is 'to square', E 'rectilinear figure' and F 'circle'. Assuming that between E and F there is only one intermediate term — that the circle becomes equal to a rectilinear figure by means of lunules — we should approximate to knowledge.

{4} When, however, BC is not more probable than AC, or there are several intermediate terms, I do not use the expression 'reduction'; nor when the proposition BC is immediate; for such a statement implies knowledge.

Book 2 **Chapter 27**

{1} A probability εικος is not the same as a sign σημειον. The former is a generally accepted premiss; for that which people know to happen or not to happen, or to be or not to be, usually in a particular way, is a probability: *e.g.*, that the envious are malevolent that those who are loved are affectionate. A sign, however, means a demonstrative premiss which is necessary or generally accepted. That which coexists with something else, or before or after whose happening something else has happened, is a sign of that something's having happened or being.

{2} An enthymeme is a syllogism from probabilities or signs; and a sign can be taken in three ways — in just as many ways as there are of taking the middle term in the several figures ...

{3} We must either classify signs in this way, and regard their middle term as an index τεκμηριον (for the name 'index' is given to that which causes to know, and the middle term is especially of this nature), or describe the arguments drawn from the extremes as 'signs', and that which is drawn from from the middle as an 'index'. For the conclusion which is reached through the first figure is most generally accepted and most true.

Aristotle: The Art of Rhetoric

Book 1 **Chapter 2**

{1} But for purposes of demonstration, real or apparent, just as Dialectic possesses two modes of argument, induction and the syllogism, real or apparent, the same is the case in Rhetoric; for the example is induction, and the enthymeme a syllogism, and the apparent enthymeme an apparent syllogism. Accordingly I call an enthymeme a rhetorical syllogism, and an example rhetorical induction.

{2} But since few of the propositions of the rhetorical syllogism are necessary, ... it is evident that the materials from which enthymemes are derived will be sometimes necessary, but for the most part only generally true; and these materials being probabilities and signs, it follows that these two elements must correspond to these two kinds of propositions, each to each. ...

Aristotle: On the Soul

Book 2 **Chapter 1**

{1} We describe one class of existing things as substance; and this we subdivide into three: (1) matter, which in itself is not an individual thing; (2) shape or form, in virtue of which individuality is directly attributed, and (3) the compound of the two. Matter is potentiality, while form is realization or actuality, and the word actuality is used in two senses, illustrated by the possession of knowledge and the exercise of it. [Knowledge is power, *i.e.*, potential.]

{2} Bodies seem to be pre-eminently substances, and most particularly those which are of natural origin; for these are the sources from which the rest are derived. But of natural bodies some have life and some have not; by life we mean the capacity for self-sustenance, growth, and decay. Every natural body, then, which possesses life must be substance, and substance of the compound type.

{3} But since it is a body of a definite kind, *viz.*, having life, the body cannot be soul, for the body is not something predicated of a subject, but rather is itself to be regarded as a subject, *i.e.*, as matter.

{4} So the soul must be substance in the sense of being the form of a natural body, which potentially has life. And substance in this sense is actuality. The soul, then, is the actuality of the kind of body we have described. But actuality has two senses, analogous to the possession of knowledge and the exercise of it.

{5} Clearly actuality in our present sense is analogous to the possession of knowledge; for both sleep and waking depend upon the presence of soul, and waking is analogous to the exercise of knowledge, sleep to its possession but not its exercise. Now in one and the same person the possession of knowledge comes first. The soul may therefore be defined as the first actuality of a natural body potentially possessing life; ...

{6} So one need no more ask whether body and soul are one than whether the wax and the impression it receives are one, or in general whether the matter of each thing is the same as that of which it is the matter; for admitting that the terms unity and being are used in many senses, the paramount sense is that of actuality.

{7} We have, then, given a general definition of what the soul is: it is substance in the sense of formula; *i.e.*, the essence of such-and-such a body [a natural body potentially possessing life].

{8} Suppose that an implement, *e.g.*, an axe, were a natural body; the substance of the axe would be that which makes it an axe, and this would be its soul; suppose this removed, and it would no longer be an axe, except equivocally. ...

{9} If the eye were a living creature, its soul would be its vision; for this is the substance in the sense of formula of the eye. But the eye is the matter of vision, and if vision fails there is no eye, except in an equivocal sense, as for instance a stone or painted eye. ...

{10} The waking state is actuality in the same sense as the cutting of the axe or the seeing of the eye, while the soul is actuality in the same sense as the faculty of the eye for seeing, or of the implement for doing its work. ...

{11} It is also uncertain whether the soul as an actuality bears the same relation to the body as the sailor to the ship.

Chapter 2

{12} But since the definite and logically more intelligible conception arises from the vague but more obvious data of sense, we must try to review the question of the soul in this light; for a definitive formula ought not merely to show the fact, as most definitions do, but to contain and exhibit the cause.

{13} But in practice the formulae of our definitions are like conclusions; for instance, what is squaring a rectangle? The construction of an equilateral rectangle equal to an oblong rectangle. Such a definition is merely a statement of the conclusion. But if a man says that squaring a rectangle is the finding of a mean proportional, he is giving the underlying cause of the thing to be defined.

Charles Sanders Peirce:

The First Rule of Reason **CP 1.135-136**

Upon this first, and in one sense this sole, rule of reason, that in order to learn you must desire to learn, and in so desiring not be satisfied with what you already incline to think, there follows one corollary which itself deserves to be inscribed upon every wall of the city of philosophy:

Do not block the way of inquiry.

Although it is better to be methodical in our investigations, and to consider the economics of research, yet there is no positive sin against logic in *trying* any theory which may come into our heads, so long as it is adopted in such a sense as to permit the investigation to go on unimpeded and undiscouraged.

Pragmatism - The Logic of Abduction **CP 5.196**

{1} If you carefully consider the question of pragmatism you will see that it is nothing else than the question of the logic of abduction. That is, pragmatism proposes a certain maxim, which, if sound, must render needless any further rule as to the admissability of hypotheses to rank as hypotheses, that is to say, as explanations of phenomena held as hopeful suggestions; and, furthermore, this is *all* that the maxim of pragmatism really pretends to do, at least so far as it is confined to logic, and is not understood as a proposition in psychology. For the maxim of pragmatism is that a conception can have no logical effect or import differing from that of a second conception except so far as, taken in connection with with other conceptions and intentions, it might conceivably modify our practical conduct differently from that second conception.

{2} Now it is indisputable that no rule of abduction would be admitted by *any* philosopher which should prohibit on any formalistic grounds any inquiry as to how we ought in consistency to shape our practical conduct. Therefore, a maxim which which looks only to possibly practical considerations will not need any supplement in order to exclude any hypotheses as inadmissable. What hypotheses it admits all philosophers would agree ought to be admitted. On the other hand, if it be true that nothing but such considerations has any logical effect or import whatever, it is plain that the maxim of pragmatism cannot cut off any kind of hypothesis which ought to be admitted. Thus, the maxim of pragmatism, if true, fully *covers* the entire logic of abduction.

{3} It remains to inquire whether this maxim may not have some *further* logical effect. If so, it must in some way affect inductive or deductive inference. But that pragmatism cannot interfere with induction is evident; because induction simply teaches us what we have to expect as a result of experimentation, and it is plain that any such expectation *may* conceivably concern practical conduct. In a certain sense it *must* affect *deduction*. Anything which gives a rule to abduction and so puts a limit upon admissable hypotheses will cut down *the premisses* of deduction, and thereby will render a *reductio ad absurdum* and other equivalent forms of deduction possible which would not otherwise have been possible.

On the Definition of Logic

from: Parts of a Carnegie Application **L75**

Version 1 **NE, vol. 4, p. 54**

Logic is *formal semiotic*. A sign is something, *A*, which brings something, *B*, its *interpretant* sign, determined or created by it, into the same sort of correspondence (or a lower implied sort) with something, *C*, its *object*, as that in which itself stands to *C*. This definition no more involves any reference to human thought than does the definition of a line as the place within which a particle lies during a lapse of time. It is from this definition that I deduce the principles of logic by mathematical reasoning ...

Version 2 **NE, vol. 4, p. 20**

Logic will here be defined as *formal semiotic*. A definition of a sign will be given which no more refers to human thought than does the definition of a line as the place which a particle occupies, part by part, during a lapse of time. Namely, a sign is something, *A*, which brings something, *B*, its *interpretant* sign determined or created by it, into the same sort of correspondence with something, *C*, its *object*, as that in which itself stands to *C*. It is from this definition, together with a definition of 'formal', that I deduce mathematically the principles of logic.

Types of Reasoning **MS 441**

NE, vol. 4, p. 184

We see three types of reasoning. The first figure embraces all Deduction whether necessary or probable. By means of it we predict the special results of the general course of things, and calculate how often they will occur in the long run. A definite probability always attaches to the Deductive conclusion because the mode of inference is necessary. The third figure is Induction by means of which we ascertain how often in the ordinary course of experience one phenomenon will be accompanied by another. No definite probability attaches to the Inductive conclusion, such as belongs to the Deductive conclusion; but we can calculate how often inductions of given structure will attain a given degree of precision. The second figure of reasoning is Retroduction. Here, not only is there no definite probability to the conclusion, but no definite probability attaches even to the mode of inference. We can only say that the Economy of Research prescribes that we should at a given stage of our inquiry try a given hypothesis, and we are to hold to it provisionally as long as the facts will permit. There is no probability about it. It is a mere suggestion which we tentatively adopt.

The Three Stages of Inquiry **CP 6.468-473**

from: A Neglected Argument for the Reality of God **P3, p. 366-370**

{1} Every inquiry whatsoever takes its rise in the observation, in one or another of the three Universes, of some surprising phenomenon, some experience which either disappoints an expectation, or breaks in upon some habit of expectation of the *inquisiturus*; and each apparent exception to this rule only confirms it. ...

The inquiry begins with pondering these phenomena in all their aspects, in the search of some point of view whence the wonder shall be resolved. At length a conjecture arises that furnishes a possible Explanation, by which I mean a syllogism exhibiting the surprising fact as necessarily consequent upon the circumstances of its of its occurrence together with the truth of the credible conjecture, as premisses. On account of this Explanation, the inquirer is led to regard his conjecture, or hypothesis, with favor. ...

The whole series of mental performances between the notice of the wonderful phenomenon and the acceptance of the hypothesis, during which the usually docile understanding seems to hold the bit between its teeth and to have us at its mercy, the search for pertinent circumstances and the laying hold of them, sometimes without our cognizance, the scrutiny of them, the dark laboring, the bursting out of the startling conjecture, the remarking of its smooth fitting to the anomaly, as it is turned back and forth like a key in a lock, and the final estimation of its Plausibility, I reckon as composing the First Stage of Inquiry. Its characteristic formula of reasoning I term Retroduction, *i.e.*, reasoning from consequent to antecedent. ...

{2} Retroduction does not afford security. The hypothesis must be tested.

This testing, to be logically valid, must honestly start, not as Retroduction starts, with scrutiny of the phenomena, but with examination of the hypothesis, and a muster of all sorts of conditional experiential consequences which would follow from its truth. This constitutes the Second Stage of Inquiry. For its characteristic form of reasoning our language has, for two centuries, been happily provided with the name Deduction. ...

{3} The purpose of Deduction, that of collecting consequents of the hypothesis, having been sufficiently carried out, the inquiry enters upon its Third Stage, that of ascertaining how far those consequents accord with Experience, and of judging accordingly whether the hypothesis is sensibly correct, or requires some inessential modification, or must be entirely rejected. Its characteristic way of reasoning is Induction. ...

Paradigm, Analogy, Example **CP 1.65, 69**

65. Analogy (Aristotle's παραδειγμα) combines the characters of Induction and Retroduction.

69. *Analogy* is the inference that a not very large collection of objects which agree in various respects may very likely agree in another respect. For instance, the earth and Mars agree in so many respects that it seems not unlikely they may agree in being inhabited.

CP 1.367, 369
CP 2.148, 513, 516, 733, 734, 787

787. Among probable inferences of mixed character, there are many forms of great importance. The most interesting, perhaps, is the argument from Analogy, in which, from a few instances of objects agreeing in a few well-defined respects, inference is made that another object, known to agree with the others in all but one of those respects, agrees in that respect also.

Appendix B: User Manual for Theme 1

Theme 1 is divided into three main operational sections:

Order serves merely as a place to begin and as a utility for accessing a pair of optional menu files.

Index operates as an empirical program, to induce a literal model of a language as encountered in actual experience.

Study operates as a reasoning program, to deduce a logical model of a universe as described in symbolic expressions.

This user manual for Theme 1 is divided into the following parts:

Part 1 provides general information about the main interactive control structure of Theme 1, the labeled prompt.

Part 2 describes the specific functions of Theme 1 in outline form, under the corresponding prompt headings.

Part 3 tells how to create propositional logic files for use in the Study module of Theme 1.

User Manual: Part 1

Labeled Prompts

Theme 1 is organized, as an outline, into a hierarchy of sections with specific functions. Each section is introduced by a labeled prompt of the form

(theme (

with the cursor resting, in dim video tones, beneath the left parenthesis at the right edge of the prompt.

At any such prompt, one of two actions is permitted:

> **Accept** the prompt by pressing the <space>, <comma>, or <enter> key. Do this if you want to execute the function named in the prompt label. Theme 1 acknowledges this response by brightening the edge of the prompt and then performs the next step of the corresponding function.
>
> **Reject** the prompt by pressing the <period> key or the <escape> key. This causes Theme 1 to skip the function named in the prompt label and to move on to the next available section of the program, or to quit the program.

Some functions will need a disk file to read from. In this case the next prompt will be of the form

(read ... file (

and Theme 1 will pause until the name of a suitable disk file is entered. Entering an empty string for the file name, i.e., just pressing the <enter> or <carriage return> key, will cause the read function to start a new file in memory. This file may at a later stage be named and saved to disk.

Other functions will need a disk file to write to. In this case the next prompt will be of the form

(write ... file (

and Theme 1 will wait until the name of a suitable disk file is entered. Just pressing the <enter> key will cause the file in memory to be written out to the screen.

User Manual: Part 2

Specific Functions

Order

(order (

Accept this prompt to begin or continue using the program. Next you will see a message reminding you to load the menu files. These must be read from disk at the next two prompts if you intend to use the Index function, but are not needed for the Study function.

Reject this prompt to quit using the program.

(read lex file (

Accept this prompt to continue using the program. Then enter "menu.lex" if you plan to use Index, or just press the <enter> key if not.

Rejecting this will return you to the Order prompt.

(read lit file (

Accept this prompt to continue using the program. Then enter "menu.lit" if you plan to use Index, or just press the <enter> key if not.

Rejecting this will return you to the Order prompt.

Index

(index (

Accept this prompt to begin using the Index function.

Reject this prompt to skip past Index to the Study section.

(read lex file (

Accept this prompt to continue with Index. Then enter the name of a previously formed lex file, or just press the <enter> key to start a new lex file.

Rejecting this will return you to the Index prompt.

(read lit file (

Accept this prompt to continue with Index. Then enter the name of a previously formed lit file, or just press the <enter> key to start a new lit file.

Rejecting this will return you to the Index prompt.

If you accept the Index prompt and supply valid (or empty) file names at both read prompts, you will have initiated the Index environment proper. The program operating in the background will be acting as a learner/inducer of the data stream language you supply from the keyboard.

The Alphabet A consists of all the printable characters on the ordinary keyboard (no control, alternate, or function keys) excluding only those that Index uses for command keys.

The keys that Index reserves for special functions are classified in the following table:

Selection	Accept	Reject
in Part	<comma> <left parens>	<period> <right parens>
in Full	<space> <enter>	<backspace> <escape>

If you start Index with an empty lit file you will see a single line display consisting of a period (to indicate a blank node of the literal tree) and a zero (indicating the frequency count for that node).

If you begin with non-empty lex and lit files you will see a screen like:

```
able                        2        1.00        0.000
 baker                       2        1.00        0.000
  can                         2        1.00        0.000
   cook                        1        0.50        0.500
    cakes                       1        1.00        0.000
```

A sequence of words from the lit file is displayed in dim video tones down the first diagonal column, with the cursor resting at the first character of the first word in the sequence.

The second column shows the incidence frequency N(i) for each word relative to the entire sequential context that precedes it in the display.

The third column gives the conditional probability or relative frequency P(i) = N(i) / N(i–1) for each word with respect to its context.

The fourth column contains the entropy contribution $-P(i)\cdot\log_2 P(i)$. Summed over all words appearing in the same context, these values yield the informational uncertainty (in bits) at that juncture in the tree.

The rules in force in this environment can be stated in somewhat concise terms, which we proceed to do in the next few pages. Afterwards, we return to take up some examples in a more discursory fashion.

The Index function embodies some general principles of organization that are useful to keep in mind. This section outlines these ideas in a global fashion, in terms of what properties are in force and what actions are available at any given time.

At any given time the program will be working at one of **two** levels:

(1) At the **Strand** level, where it begins by default, the program is operating with strands treated as sequences of words.

(2) At the **Word** level, which is entered whenever you type some ordinary character, the program is dealing with words considered as sequences of alphabet characters.

At any time the screen is divided into **three** distinct regions:

(1) **Past**: in normal video tones, above and to the left of the cursor.

(2) **Present**: in dim video, either at or on a line with the cursor.

(3) **Prospective**: in dim video, to the right of and below the cursor.

At any time there are **four** kinds of keys to type:

(1) **Accept** keys <comma> and <left parens>. Pressing one of these will accept the present item of the displayed sequence and move the cursor on to the next item.

  (a) At the **Strand** level this means accepting the present word and moving on to a prospective subsequent word.

  (b) At the **Word** level this serves to accept the present letter of a displayed word.

(2) **Reject** keys: in general these keys act to exchange a current display for an alternate one, and they come in several varieties:

  (a) The <period> key does partial rejects. It keeps fixed the past part of the displayed sequence and points the “attention” of the program to the next in

a cycle of alternate presents, at each point projecting a new extension of the sequence.

(i) At the **Strand** level this serves to cycle through all the sequels of a word, viewing a typical prospect of each.

(ii) At the **Word** level this allows you to view a selection of words having a given prefix.

These operators override the typicality effect, leaving word and path associations wherever they were last "pointed", and where they will remain until the files are sorted.

To leave the search cycle, it will be necessary to type either a backspace or one of the accept keys.

(b) The <tab> key, when pressed after accepting some word, will cycle through all occurrences of that word in the index that were present in the index files at load time.

To leave the <tab> search it is necessary to type some other (non-tab) key.

(c) The <backspace>, <control-x>, <control-z>, and <escape> keys all perform total rejects of the current word or list.

The <escape> key, in addition, is used to exit the interior phase of the index procedure.

(3) **Default** keys: the <space> and <enter> keys. These keys serve as total accepts of the displayed sequence, either word or list. This causes the frequency counts associated with that sequence to be incremented, in effect "reinforcing" that path or word in the "experience" of the index usage data base.

(4) **Ordinary** keys: all the usual characters, that one is forced to type when entering words or paths not already in the index.

Typing an ordinary character shifts the program to the word level and displays a typical word beginning with that letter.

It is always permissible to keep typing over the projected word, with the program continuing to guess a word on each new prefix, but if at any stage the word you intend is already displayed, then you may simply press <space> or <enter> to accept it.

Alternately, you may press <comma> repeatedly to accept some initial segment of the displayed word.

To accept just some prefix of a longer word (e.g., to select the prefix "at" when the word "attention" is displayed), press <comma> repeatedly up through the end of the desired word, then press <period> repeatedly until a terminal blank appears, then press <space> or <enter>.

Study

(study (

Accept this prompt to begin using the Study function.

Reject this prompt to skip past Study to the Order prompt.

(read lex file (

Accept this prompt to continue with Study. Then enter the name of a previously formed lex file, or just press <enter> to start with a null lex file.

You will not be able to do much in this section without a lex file already made up.

Rejecting this will return you to the Study prompt.

(read log file (

Accept this prompt to continue with Study. Then enter the name of a previously formed log file, or just press <enter> to start with a null log file.

You will not be able to do much in this section without a log file already made up.

Rejecting this will bring up the optional prompt that is discussed next.

(read num file (

Accept this prompt, which you will only see after rejecting the log file read prompt, if you want to read the *numbered* form of a log file. The num file is just a variant form of the log file that is created by one of the Study functions from a previously formed log file. Num files are intended for use in a future version of the program and will not be further discussed here. Therefore we recommend that you always:

Reject this prompt, thereby returning to the Study prompt, and try again to read the lex and log files.

User Manual: Part 3

Creating Log Files

In order to create log files for Study you will need a plain ASCII editor or word processor, one which uses the ASCII <carriage return><line feed> combination (#13#10 or ^M^J) for its <end of line> sequence, and which uses the ASCII <control-z> (#26 or ^Z) for its <end of file> character. The Turbo Pascal editor ((c) Borland International) works just fine.

In order to use a particular log file with Study you will also need to have prepared a lex file containing all the words in that log file. You do this by using Index, simply typing in all the words you need in any convenient arrangement, then saving the lex and lit files thus created. (You do not need to keep the lit file for Study, but it helps to remember what words were stored in the lex file.)

With this preparation, use your text editor and the syntax for Ex to create a text file bearing the proposition to be Studied. (It helps to give it a file name with a "log" suffix or an "egN" suffix, N being some digit, if you are using several log file examples with the same lex file.) Refer to Part 2 for the mechanics of loading and saving files in the Index and Study sections of Theme 1.